GRASP/Ada

Graphical Representations of Algorithms, Structures, and Processes for Ada

The Development of a Program Analysis Environment for Ada

Reverse Engineering Tools For Ada

Task 2, Phase 3 Six-Month Report

Contract Number NASA-NCC8-14

Department of Computer Science and Engineering
Auburn University, AL 36849-5347

Contact: James H. Cross II, Ph.D.
Principal Investigator
(205) 844-4330

February 1991

ACKNOWLEDGEMENTS

We appreciate the assistance provided by NASA personnel, especially Mr. Keith Shackelford whose guidance has been of great value. Portions of this report were contributed by each of the members of the project team. The following is an alphabetical listing of the project team members.

Faculty Investigator:

Dr. James H. Cross II, Principal Investigator

Graduate Research Assistants:

Richard A. Davis
Charles H. May
Kelly I. Morrison
Timothy A. Plunkett
Narayana S. Rekapalli
Darren Tola

The following trademarks are referenced in the text of this report.

Ada is a trademark of the United States Government, Ada Joint Program Office.

Software through Pictures (StP), **Ada Development Environment (ADE)**, and **IDE** are trademarks of Interactive Development Environments.

PostScript is a trademark of Adobe Systems, Inc.

VAX and **VMS** are trademarks of Digital Equipment Corporation.

VERDIX and **VADS** are trademarks of Verdix Corporation.

UNIX is a trademark of AT&T.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 Introduction and Executive Summary

Computer professionals have long promoted the idea that graphical representations of software can be extremely useful as comprehension aids when used to supplement textual descriptions and specifications of software, especially for large complex systems. The general goal of this research is the investigation, formulation and generation of *graphical representations of algorithms, structures, and processes for Ada* (GRASP/Ada). The present task, in which we describe and categorize various graphical representations that can be extracted or generated from source code, is focused on *reverse engineering*.

Reverse engineering normally includes the processing of source code to extract higher levels of abstraction for both data and processes. Our primary motivation for reverse engineering is increased support for software reusability, verification, and software maintenance, all of which should be greatly facilitated by automatically generating a set of "formalized diagrams" to supplement the source code and other forms of existing documentation. For example, Selby [SEL85] found that code reading was the most cost effective method of detecting errors during the verification process when compared to functional testing and structural testing. And Standish [STA85] reported that program understanding may represent as much as 90% of the cost of maintenance. Hence, improved comprehension efficiency resulting from the integration of graphical notations and source code could have a significant impact on the overall cost of software production. The overall goal of the GRASP/Ada project is to provide the foundation for a CASE (computer-aided software engineering) environment in which reverse engineering and forward engineering (development) are tightly coupled. In this environment, the user may specify the software

in a graphically-oriented language and then automatically generate the corresponding Ada code [ADA83]. Alternatively, the user may specify the software in Ada or Ada/PDL and then automatically generate the graphical representations either dynamically as the code is entered or as a form of post-processing. Appendix A contains a comprehensive taxonomy of reverse engineering, including definitions of terms.

Figure 1 provides an overview to the three phases of the GRASP/Ada project. Ada source code or PDL is depicted as the basic starting point for the GRASP/Ada toolset. Each phase is briefly described below. Phases 1 and 2 of GRASP/Ada have been completed and a new graphical notation, the Control Structure Diagram (CSD) for Ada and the supporting software tool is now being prepared for evaluation [CRO88, CRO89, CRO90a-d]. In Phase 3, the focus is on a subset of Architectural Diagrams that can be generated automatically from source code with the CSD included for completeness. These are described briefly in the order that they might be generated in a typical reverse engineering scenario.

1.1 Phase 1 - The Control Structure Diagram For Ada

Phase 1 concentrated on a survey of graphical notations for software and the development of a new algorithmic or PDL/code level diagram for Ada. Tentative graphical control constructs for the *Control Structure Diagram* (CSD) were created and initially prototyped in a VAX/VMS environment. This included the development of special diagramming fonts for both the screen and printer and the development of parser and scanner using UNIX based tools such as LEX and YACC. Appendix B provides a detailed description of the CSD and the rationale for its development. The final report for Phase 1 [CRO89] contains a complete description of all accomplishments of Phase 1.

Figure 1. GRASP/Ada Overview

1.2 Phase 2 - The GRASP/Ada Prototype and User Interface

During Phase 2, the prototype was extended and ported to a Sun/UNIX environment. The development of a user interface based on the X Window System represented a major part of the extension effort. Verdix Ada and the Verdix DIANA interface were acquired as potential commercial tools upon which to base the GRASP/Ada prototype. Architectural diagrams for Ada were surveyed and the OOSD notation [WAS89] was identified as having the best potential for accurately representing many of the varied architectural features of an Ada software system. Phase 2 also included the preliminary design and a separate prototype for an architectural CSD. The best aspects of architectural CSD are expected to be integrated into the prototype during Phase 3. The final report for Phase 2 [CRO90c] contains a complete description of the accomplishments of Phase 2.

1.3 Phase 3 - Integration, Evaluation and Release

Phase 3 has two major thrusts: (1) completion and release of an operational GRASP/Ada prototype which generates CSDs and (2) the analysis, design and development of a preliminary prototype which generates object diagrams directly from Ada source code. Completion of the GRASP/Ada CSD prototype includes the development of an intermediate representation of the CSD to increase efficiency and provide for extensibility. A major subtask of Phase 3 is preparing the prototype for release to the research community, business and industry. To date, over 80 requests for information regarding GRASP/Ada have been received as a result of publications generated from this research. Responding to these requests are an important element of the ongoing evaluation and refinement of the GRASP/Ada reverse engineering system.

The development of a preliminary prototype for generating architectural object diagrams (ODgen) for Ada source/PDL is an effort to determine feasibility rather than deliver an operational prototype as above. Research has indicated that the major obstacle for automatic object diagram generation is the automatic layout of the diagrams in a human readable and/or aesthetically pleasing format. A user extensible rule base, which automates the diagram layout task, is currently being formulated. If a satisfactory solution to the layout problem can be found, the development of the components to recover the information to be included in the diagram, although a major effort, is expected to be fairly straightforward. Interactive Development Environment's Software through Pictures (IDE/StP), which supports the OOSD notation in a forward engineering sense, has been acquired as a candidate for a commercial CASE environment with which to integrate GRASP/Ada reverse engineering system.

2.0 The System Model

The general system model for the GRASP/Ada prototype is described in this section. The overall functionality of the system is briefly described from a data flow perspective and then each of the GRASP/Ada components is presented in the form of a system block diagram.

2.1 GRASP/Ada System Data Flow

Figures 2 and 3 describe the major processes and overall flow of information within the GRASP/Ada system. The primary input is Ada source code GRASP commands and the primary outputs are control structure diagrams, object diagrams and library information. The Ada source code is assumed to be syntactically correct.

2.2 GRASP/Ada System Block Diagram

Figure 4 depicts the major system components hierarchically to illustrate the layers and component interfaces. The user interface (not shown in the system data flow diagram) was built using the X Window System and provides general control and coordination among the other components.

The control structure diagram generator, **CSDgen**, has its own parser/scanner built using FLEX and BISON, successors of LEX and YACC. It also includes its own printer utilities. As such, CSDgen is a self-sufficient component which can be used from the user interface or the command line without the commercial components required by the object diagram generation component. A CSD editor, **CSDedit** (not shown), is currently in the planning stages. It will provide editing capabilities for directly modifying the CSD produced

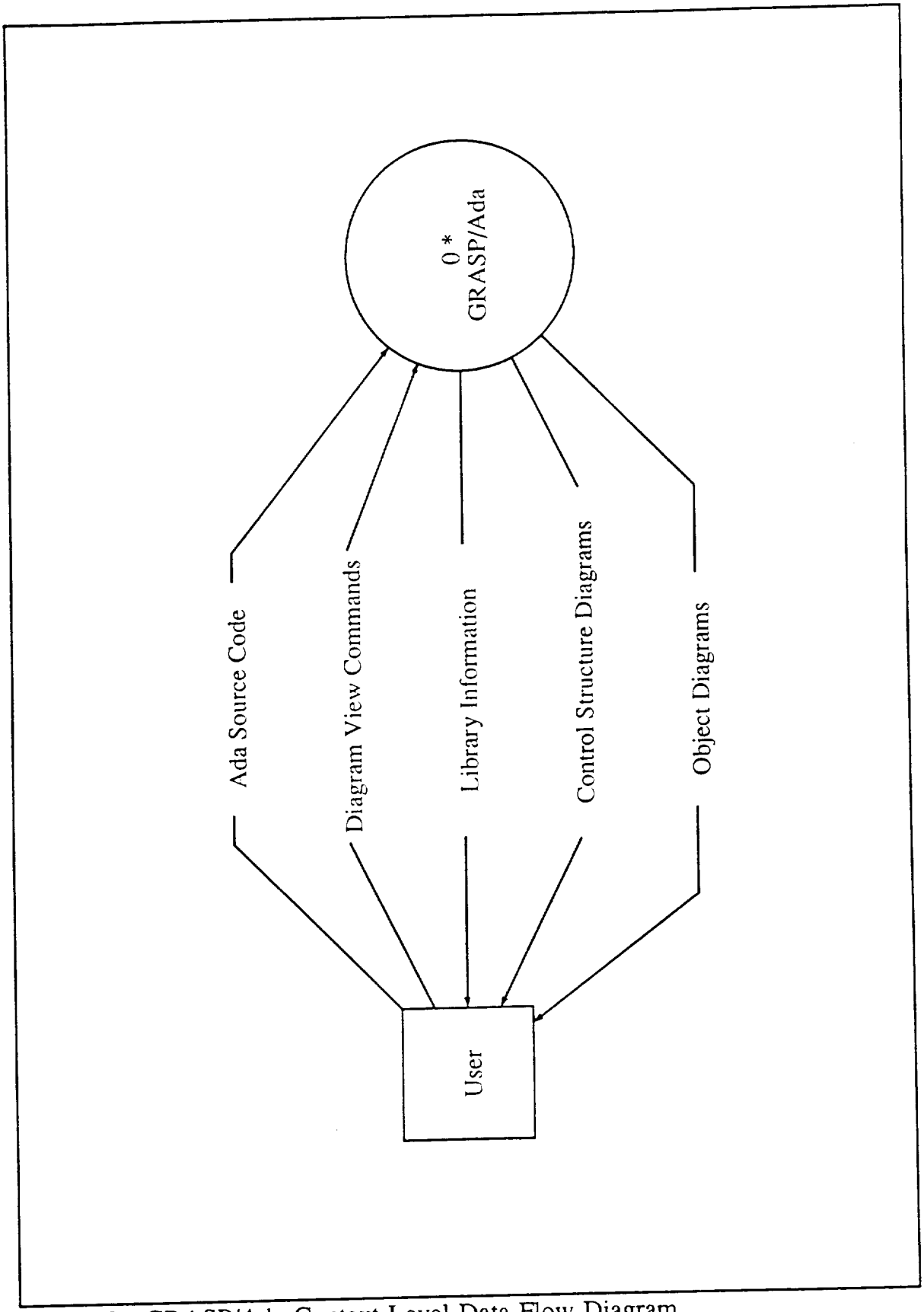


Figure 2. GRASP/Ada Context Level Data Flow Diagram

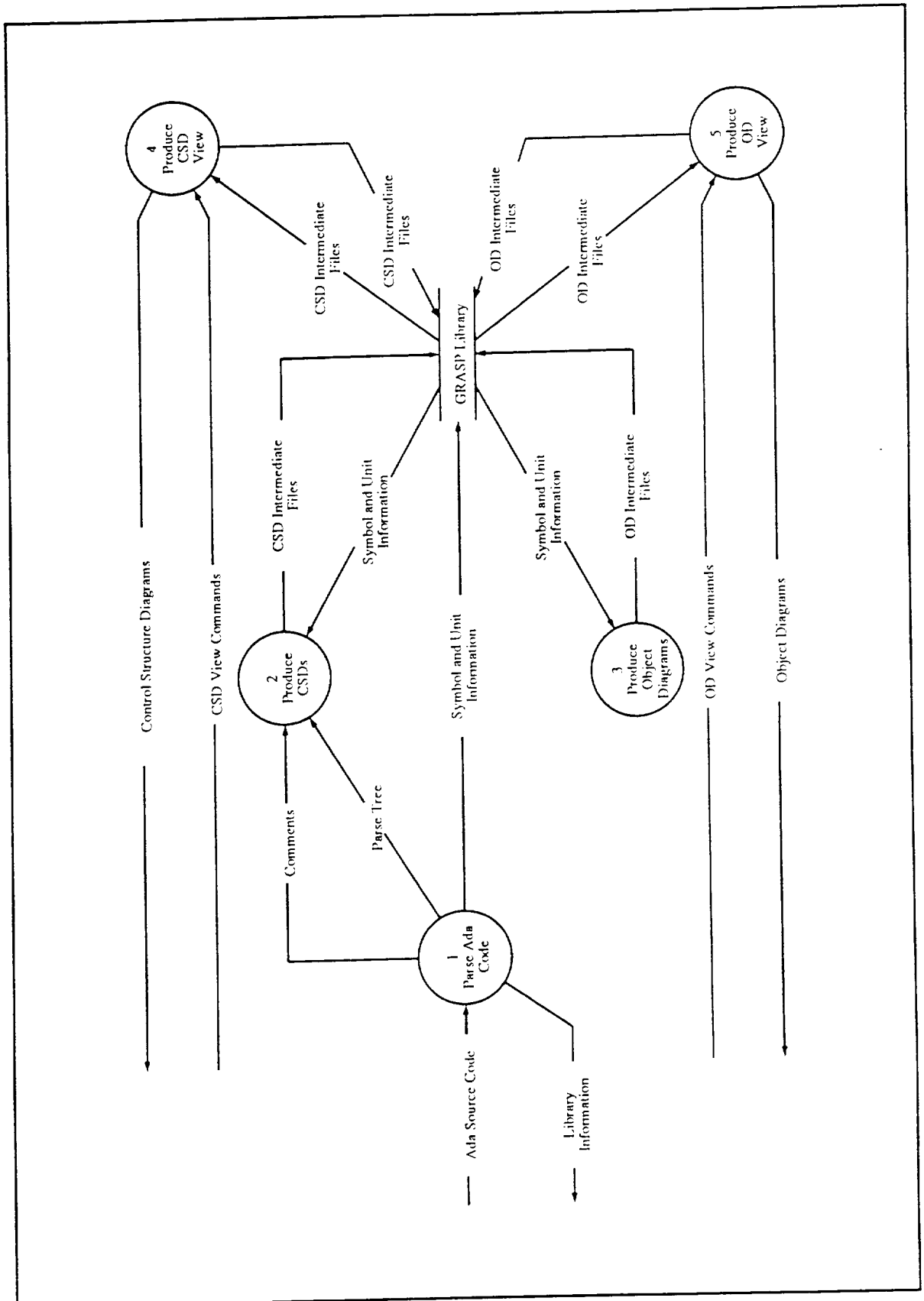


Figure 3. GRASP/Ada System Level Data Flow Diagram

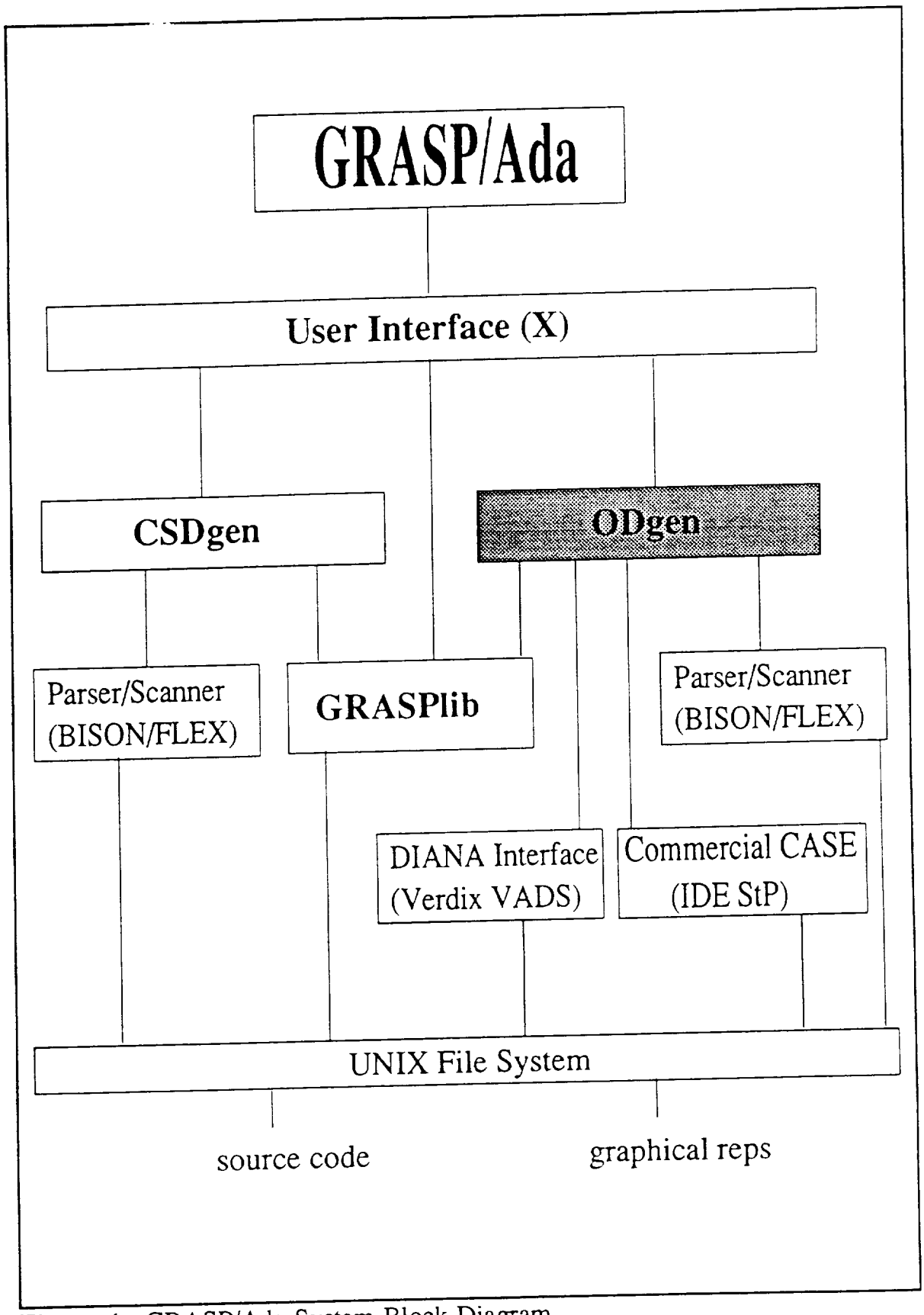


Figure 4. GRASP/Ada System Block Diagram

by CSDgen. Without CSDedit, changes must be made to the source code and then the CSD must be regenerated.

The object diagram generation component, **ODgen**, is in the analysis phase. The feasibility of automatic diagram layout is currently under investigation. Beyond automatic diagram layout, several design alternatives are being investigated. The major alternatives include the decision of whether to attempt to integrate GRASP/Ada directly with commercial components, namely (1) the Verdix Ada development system (VADS) and DIANA interface for extraction of diagram information and (2) IDE's Software through Pictures, Ada Development Environment (IDE/StP/ADE) for the display of the object diagrams. Each of these components are indicated in Figure 4.

The GRASP/Ada library component, **GRASPlib**, provides for coordination of all generated items with their associated source code. This facilitates navigation among the diagrams and the production of sets of diagrams. Both CSDgen and ODgen produce library entries as Ada source is processed.

In the following sections, the general functional requirements and prototype implementation (in progress) are described for each of the major GRASP/Ada components: the control structure diagram generator, the object diagram generator, and the user interface.

3.0 Control Structure Diagram Generator

The GRASP/Ada control structure diagram generator (CSDgen) is described in this section. The detailed specifications and current issues are described below. Examples of the CSD are presented in conjunction with the User Interface in Section 5.0 and in Appendix C. The rationale for the development of the CSD, which has been detailed in previous reports [CRO89, CRO90c], is summarized in Appendix B.

3.1 Generating the CSD

The primary function of CSDgen is to produce a CSD for a corresponding Ada source file. CSDgen has its own parser/scanner constructed using LEX/YACC based software tools available with UNIX. Although a complete parse is done during CSD generation, CSDgen assumes the Ada source code has been previously compiled and thus is syntactically correct. Currently, little error recovery is attempted when a syntax error is encountered. The diagram is simply generated down to the point of the error. The Version 1 and Version 2 CSDgen prototypes built the diagram directly during the parse by inserting CSD graphics characters into a file along with text. To increase efficiency and improve extensibility, the Version 3 CSDgen prototype will use an intermediate representation which is described below.

Since GRASP/Ada is expected to be used to process and analyze large existing Ada software systems consisting of perhaps hundreds of files, an option to generate all the CSDs at once is required. Generating a set of CSDs can be facilitated by entering *.*a* or some other wildcard with a conventional source file extension, for the file name. A CSD generation **summary window** should display the progress of the generation by listing each

file as it is being processed and any resulting error messages. The summary should conclude with number of files processed and the number of errors encountered. The default for each CSD file name is the source file name with **.csd** appended. As the CSDs are generated, the GRASP library is updated. Generating a set of CSDs can be considered a user interface requirement rather than strictly a CSD generator requirement.

3.2 Displaying the CSD - Screen and Printer

Basic display capabilities to the screen and printer were implemented during Phase 2. Screen display is facilitated by sending the CSD file to a CSD window opened under an X Window manager. Printing is accomplished by converting the CSD file to a PostScript file and then sending it to a printer. Moving to an intermediate representation during Phase 3 will necessitate the development of a new set of display routines which will be X Window System based. However, these new routines will increase the flexibility and capability of CSDgen, thus making it more immediately useful to the research community. Layout/spacing, collapsing the CSD, and screen and printer fonts are considered below.

Layout/Spacing. The general layout of the CSD is highly structured by design. However, the user should have control over such attributes as horizontal and vertical spacing and the optional use of some diagramming symbols. In the Version 2 CSDgen prototype, horizontal and vertical spacing options were a part of the CSD file generation. In order to change these options (e.g., from single to double spacing), the CSD file had to be regenerated. In the Version 3 prototype, these options will be handled by the new display routines and, as such, can be modified dynamically without regenerating the CSD file.

Vertical spacing options include single, double, and triple spacing (default is single). Margins have been roughly controlled by the character line length selected, either 80 or 132

characters per line (default is 80). Indentation of the CSD constructs has been a constant three blank characters. Support for variable margins and indentation are being investigated in conjunction with the new display routines. In addition, several display options involving CSD graphical constructs are under consideration. For example, the boxes drawn around procedure and task entry calls may be optionally suppressed to make the diagram more compact.

Collapsing the CSD. The CSD window should provide the user with the capability to collapse the CSD based on all control constructs as well as complete diagram entities (e.g., procedures, functions, tasks and packages). This capability directly combines the ideas of chunking with control flow which are major aids to comprehension of software. An *architectural CSD* (ArchCSD) [DAV90] can be facilitated by collapsing the CSD based on procedure, function, and task entry calls, and the control constructs that directly affect these calls. The initial ArchCSD prototype was completely separate from CSDgen and required complete regeneration of the ArchCSD file for each option. In the Version 3 prototype, the ArchCSD will be generated by the display routines from the single intermediate representation of the CSD.

CSD Screen Fonts. The CSD screen font is a bitmap 14 point Courier to which the CSD graphic characters were added. The font was defined as a bitmap distribution font (BDF) then converted to SNF format required by the X Window System. Additional screen fonts may be developed as required.

CSD Printer Fonts. CSD Printer fonts were initially developed for the HP LaserJet series. These were then implemented as PostScript type 3 fonts and all subsequent font development has been directed towards the PostScript font. The PostScript font provides the most flexibility since its size is user selectable from 1 to 300 points.

Color. Although color options were briefly investigated for both the screen and printer, it was decided that they will not be pursued in the Version 3 prototype.

3.3 Navigating Through Large CSDs - Alternatives

Index (or Table of Contents). An index, similar to that presented in the Xman application provided with the X Window System for viewing manual pages, is used to navigate among a system of CSDs. The user clicks on the index entry and the corresponding CSD is displayed. The index entries would be created as CSDs are generated and stored in the GRASP/Ada library. Entries in the library are to include procedures, functions, tasks, task entries, and packages. See Section 6 below for details.

Direct Navigation Via CSD. The user is allowed to click on procedure, function, and task entry calls in the CSD directly and a separate CSD window is opened containing the selected CSD or fragment thereof. Two potential problems have been identified with this approach. Using the mouse for selection may conflict with established editing functions supported by the mouse. In addition, it may be difficult to relate the characters or CSD graphical construct with subprogram and entry names. The availability of middle mouse button for this purpose is being investigated.

3.4 Printing An Entire Set of CSDs

Printing an entire set of CSDs in an organized and efficient manner is an important capability when considering the typically large size of Ada software systems. A book format is under consideration which would include a table of contents and/or index. In the event GRASP/Ada is integrated with IDE/StP/ADE, the StP Document Preparation System could possibly be utilized for this function.

3.5 Incremental Changes to the CSD

In the present prototype, there is no capability for editing or incrementally modifying the CSD. The source code is modified using a text editor and then the CSD is regenerated. While this has been sufficient for early prototyping, especially for small programs, editing capabilities are desirable in an operational setting. An editor has been proposed and is briefly discussed in Section 7.0 Future Requirements.

3.6 Internal Representation of the CSD - Alternatives

Several alternatives are under consideration for the internal representation of the CSD in the Version 3 prototype. Each has its own merits with respect to processing and storage efficiency and is briefly described below.

Single ASCII File with CSD Characters and Text Combined. This is the most direct approach and is currently used in the version 2 prototype. The primary advantage of this approach is that combining the CSD characters with text in a single file eliminates the need for elaborate transformation and thus enables the rapid implementation of prototypes as was the case in the previous phases of this project. The major disadvantages of this approach are that it does not lend itself to incremental changes during editing and the CSD characters have to be stripped out if the user wants to send the file to a compiler.

Separate ASCII Files for CSD Characters and Text. In this approach, the file containing the CSD characters along with placement information would be "merged" with the prettyprinted source file. The primary advantage of the this approach is that the CSD characters would not have to be stripped out if the user wants to send the file to a compiler. The major disadvantage of this approach is that coordinating the two files would add complexity to generation and editing routines with little or no benefit. As a result, this

approach would be more difficult to implement than the single file approach and not provide the advantages of the next alternative.

Single ASCII File Without Hard-coded CSD Characters. This approach represents a compromise between the previous two. While it uses a single file, only "begin construct" and "end construct" codes are actually required for each CSD graphical construct in the CSD file rather than all CSD graphics characters that compose the diagram. In particular, no continuation characters would be included in the file. These would be generated by the screen display and print routines as required. The advantages of this approach would be most beneficial in an editing mode since the diagram could grow and shrink automatically as additional text/source code is inserted into the diagram. The extent of required modifications to text edit windows must be considered with this alternative.

Direct Generation From DIANA Net. If tight coupling and integration with a commercial Ada development system such as Verdix VADS is desired, then direct generation of the CSD from the DIANA net produced as a result of compilation could be performed. This would require a layer of software which traverses the DIANA net and calls the appropriate CSD primitives as control nodes are encountered. This approach would apparently eliminate the possibility of directly editing the CSD since the DIANA interface does not support modifying the net, only reading it.

3.7 Additional CSD Constructs

Task Entry and Task Exit Symbols, Label and GOTO Label Symbols. These are needed to differentiate among a task exit, function return, and goto statement, and between a task entry and label symbol.

Generic Task and Package. Dashed task and package symbols should be used to distinguish between generic and non-generic tasks and between generic and non-generic packages.

Function Call. A CSD symbol similar to that used for procedure calls should be used for function calls for consistency.

Task Entry Call. Currently the task entry symbol is the same as the task definition symbol (open-ended parallelogram). However, a call to a task entry block is similar semantically to a procedure call. Hence, it would be more appropriate to use the procedure symbol for the task entry call in the calling subprogram and the task entry itself in the task.

4.0 Object Oriented Design Diagram Generator

The object-oriented design diagram generator, or simply object diagram generator (ODgen), produces object diagrams (ODs) for a corresponding set of Ada source files. The detailed specifications and current issues are described below. A preliminary prototype is expected to be constructed to determine several of the feasibility issues.

4.1 ODgen Symbol Set

The OOSD notation [WAS89] has been selected as a basis for the Object Diagram generator (ODgen). The complete set, which was designed with the intention of using it in forward engineering, is illustrated in Figure 5. In this section, the feasibility of deriving each of these symbols during a reverse engineering effort is considered, and the modifications or supplements needed to render them suitable for the ODgen project are discussed.

Lexical Inclusion of Data Modules. The inclusion of a data module into another module may be determined from a parse of the Ada source code. If a data module is considered to be a component which contains no executable statements other than initializations, then there should be no difficulty in recognizing these modules, and their inclusion in an OD should cause no problems.

Iterative Calls to Library Modules. Again, this information may be extracted from a parse of the Ada source code. There should be no difficulty in producing an OD representation for iterative calls to library modules; however, the composition of this situation with others, such as conditional module calls, may require further analysis.

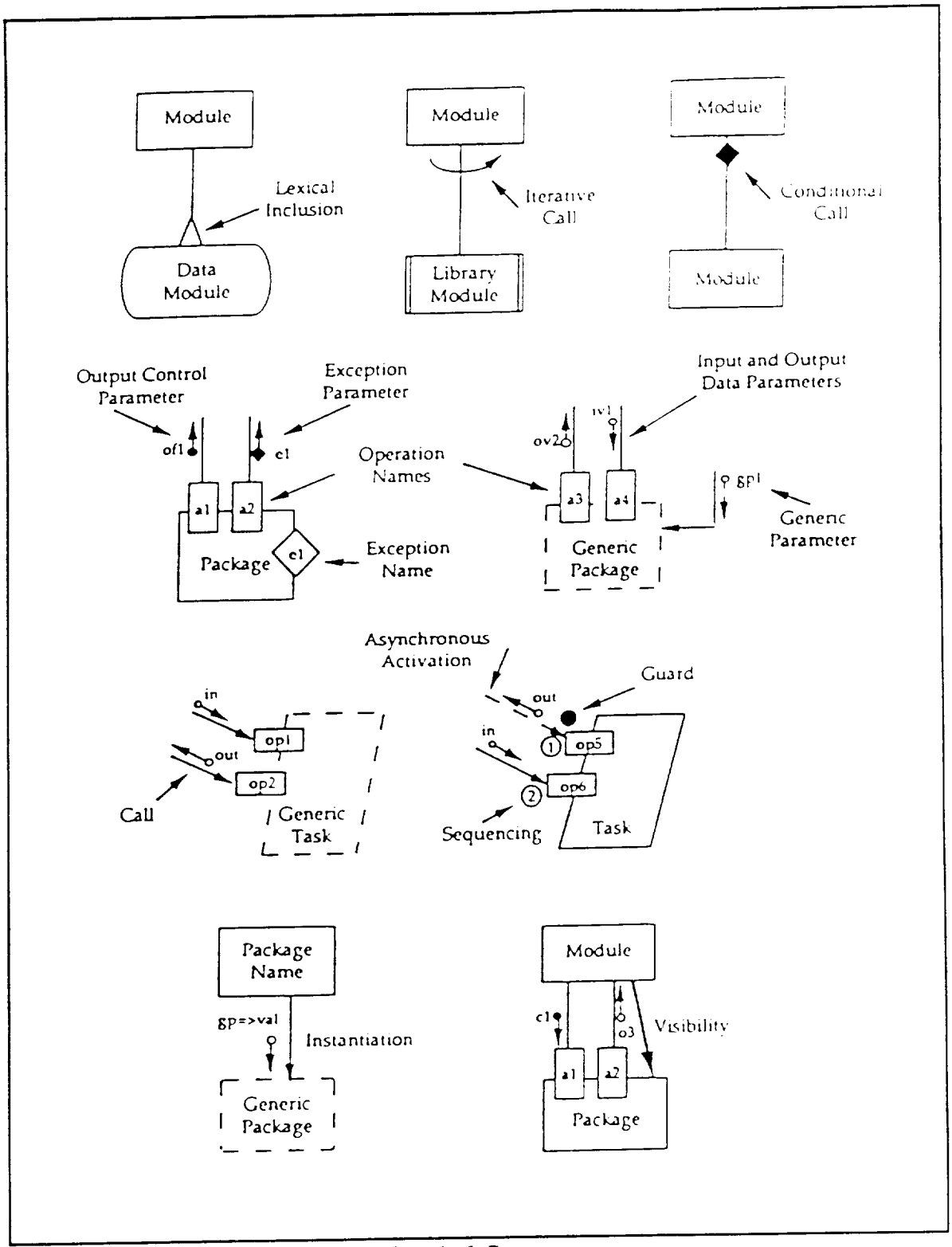


Figure 5. The OOSD Notation Symbol Set
(from *Introduction to StP OOSD Graphical Editor*, IDE, 1989, p. 59)

Conditional Module Calls. A conditional call of one module from another can be recognized during parsing, but the generation of an OD representation may prove difficult should the conditional call be composed with another type of call. For example, a program loop may conditionally call another module within the loop's body. How should this be represented in the OD? Certainly the call is a conditional one and may be represented using the conditional module call construct. However, the module is being called repetitively within a loop, so it may just as well be represented using the iterative call construct. Another possibility is to represent the call using a composition of the two representations, indicating that the module is called both iteratively and conditionally. The problem is that this raises ambiguity in that the diagram does not indicate whether the call was made conditionally in the body of a loop, or whether it was made iteratively as the consequence of some condition being true. This ambiguity must be resolved if the iterative and module call representations are to be used properly in the OD.

Package Specifications. A package may be recognized from a parse of an Ada program, and the operations contained within the package may be recognized just as easily. The direction of the parameters may also be determined syntactically through the presence of the **in**, **out**, and **in out** parameter designators. However, the distinction of parameters as either control or data parameters may not be recognized as easily. In fact, it is possible for parameters to be used as both control and data parameters, so the automated classification of an operations's parameters as control or data may not be feasible. Finally, the detection of exceptions may be determined easily through syntactic analysis.

Generic Packages. The specification of a generic package may be recognized easily from a parse of an Ada program, and the generic parameters which must be specified in an instantiation of the package, the operations provided by the package, the parameters to the

operations and their direction may also be recognized syntactically. However, the generic package suffers from the same problem as the package in the area of detection of control and data parameters. Again, the automated classification of parameters as either control or data parameters may not be feasible.

Tasks. The declaration of a task may be recognized syntactically in a parse of an Ada program. Much of the desired information needed in the creation of an OD representation of a task may also be obtained from syntactic analysis, such as the entries provided by the task, the parameters and their associated directions for each of the task entries, and any guards placed on the task entries. However, there are two items in the OOSD depiction of a task that may not be obtainable in an automated fashion during reverse engineering. The first of these is the omnipresent problem of distinguishing between control and data parameters which has already been discussed in previous paragraphs. The second is the placement of sequencing numbers on the task entries. Only in the most trivial cases may these numbers be properly derived. In more complex cases, the sequencing numbers would be meaningless or even misleading, and the OD would probably be better off by omitting these numbers.

Generic Tasks. The depiction of a generic task in the OD suffers from many of the same problems as the depiction of a task, and the reader is referred to the previous paragraphs for a discussion of these problems. Other than that, the detection and representation of a generic task should provide no further problems.

Instantiation of Generic Packages. The instantiation of a generic package in an Ada program may easily be determined syntactically. The generation of a proper OOSD symbol for generic package instantiation will require actual parameters to be matched with formal parameters. Otherwise, it should pose no difficulty.

Visibility. The depiction of the semantic visibility of a package to a module in an Ada program may be determined syntactically, but the representation may prove to be misleading. There are two "varieties" of visibility that must be represented: packages lexically included in the declarative section of the current compilation unit and packages included via the **with** clause, which are separate compilation units. For example, a package in an Ada program may only be visible to a small section of a module (for example, a block in a module containing a loop may declare the package in the declaration area and call a function in the package iteratively during the loop. The package would therefore be visible throughout the scope of the block, but would not be visible in the statements preceding and following the block. Therefore, the depiction of the package as being visible to the module could be misleading to the user unfamiliar with the underlying code. Although generating the representation is not difficult, the sensibility of utilizing the representation must be considered. When visibility is determined by the **with** clause, a separate icon is, of course, necessary and appropriate.

Symbol Interconnections and Diagram Layout. The actual automatic layout of the generated object diagram with respect to symbols and interconnections is the most formidable problem that must be solved. Whereas the CSD has a flexible but well-defined physical layout, the OD layout is not well-defined. In fact, the CASE tools that support the OOSD notation require the users to "manually" arrange the symbols. Determining the feasibility of an algorithmic and/or heuristic solution which yields a reasonably comprehensible diagram layout, and then demonstrating it, is a key component of Phase 3.

4.2 GRASP/Ada ODgen Processing Alternatives

In the development of the ODgen design specification, three distinct development methods were considered. The major difference among these methods is linked to the degree of involvement of other commercially available tools and the ability of the user to specify these tools. The first method considered was to create ODgen as a stand-alone system. A second alternative was to use GRASP/Ada as a driver for a set of subprogram invocations which would use VADS, ODgen, and StP/ADE in sequence to produce the architectural diagrams. Finally, the third alternative considered was to use GRASP/Ada as a shell from which the user could invoke each of the three tools at his convenience. In this section, these three methods are examined in more detail, and the advantages and disadvantages associated with each method are outlined.

ODgen Is Independent of Commercial Tools. This method would involve the development of a stand-alone architectural diagram generator. The generator would not be dependent on commercial tools such as VADS and StP/ADE. Instead, the parser/scanner developed in Phases I and II of the GRASP/Ada research project would be extended to extract the information needed for the representation of architectural diagrams. A method for specifying or identifying the complete set of files comprising the Ada system would have to be developed (this may require some involvement from the user). The major advantage of this method is that the tool would not be subject to the whims of the manufacturers of commercial tools (i.e., the tool would not be rendered useless if VADS were to become unsupported, if the DIANA representation were subjected to large-scale change, if the StP/ADE file formats and representation methods were to be changed, etc.). On the other hand, this method would involve substantially longer development time, as a tool for identifying the dependencies among a set of Ada source files would have to be developed.

In addition, a tool for viewing and printing the architectural diagrams would need to be developed. Because a substantial amount of effort has already been spent in the development of the GRASP/Ada X11R4 interface, extending this interface to display the architectural diagrams could benefit from the groundwork already laid in Phases I and II. The major goals which would need to be accomplished are the development of X11R4 widgets for the representation of each of the OOSD symbols, and the development of layout heuristics and modified layout widgets suitable for displaying the OOSD symbols.

ODgen Invokes VADS and StP/ADE. In this method, the ODgen component of GRASP/Ada would first invoke VADS to generate a DIANA net for the specified set of Ada source files. ODgen would then traverse this net to obtain the required information and generate an internal representation for the architectural diagrams. This information would then be shaped into a format suitable for StP/ADE and saved. Finally, StP/ADE would be invoked to view the architectural diagram. All of this would be transparent to the user: after specifying the Ada source files and a number of ODgen options, GRASP/Ada would invoke the tools in sequence and bring up StP/ADE as a subprocess displaying the generated diagrams. The major advantage in this approach is that it would utilize already-existing tools to speed the development effort. Instead of writing yet another Ada parser, intermediate representation generator and OOSD diagram displayer, the research effort could concentrate on the task of obtaining architectural details and composing meaningful architectural diagrams from them. However, relying on commercial tools could be dangerous as subtle changes in the formats of either the VADS representation or the StP/ADE representation could require major, sweeping changes in the ODgen system. In addition, the use of commercial tools could greatly limit the number of potential users for the ODgen system. Instead of only needing the ODgen system, the user would also need the VADS Ada compiler and the

StP/ADE software development system - two costly components. For many university research installations, the costs of these systems would be prohibitive and would virtually eliminate the potential use of ODgen.

GRASP Runs Independently of VADS and StP/ADE. The user invokes VADS to create DIANA nets, invokes GRASP to generate CSDs and ODs, and invokes StP/ADE to view the ODs. In this scenario, the GRASP/Ada interface would be partially customizable by the user. Instead of relying on a specific Ada intermediate representation generator and OOSD diagram displayer, the user would be able to select from a limited number of commercial tools. To accomplish this, a minimal ODgen interface for each tool would be identified and a suitable data representation would be specified. ODgen would then be designed to transform the input Ada source data into an architectural diagram representation in the output format. Then, customizing GRASP/Ada for new intermediate Ada representations and OOSD diagram formats would consist of simply writing a filter transforming the data from one representation to another. For example, customizing GRASP/Ada to work with the VADS DIANA representation would require a filter to be written to traverse the DIANA nets and store the needed architectural information into a file in ODgen's input format. Similarly, customizing GRASP/Ada to work with the StP/ADE tool would require a filter to be written translating the ODgen output format into StP/ADE's input format. This method would allow GRASP/Ada to be fairly portable without depending on strict reliance on commercially available tools. On the other hand, this method would require an extensive and easily translatable interface format to be developed for both ODgen's input and output formats. Finally, the amount of effort required for the writing of filters for new representations could be potentially quite large, depending on the format and accessibility of the new representations.

4.3 Displaying the OD - Screen and Printer

Generating visual displays of the object diagrams will require display methods to be generated for the screen and printer. Since the GRASP/Ada interface for Phases I and II was developed using the X Window System (a portable graphical environment gaining widespread acceptance) and numerous utilities have been developed in the creation of that interface, the development of a display mechanism for the object diagrams in X11R4 would be a logical extension to the previous work. In addition, the PostScript page description language was used in Phases I and II for the hardcopy output of the CSD diagrams. Because PostScript is a nearly universal output description language for laser printers, the development of PostScript utilities for printing GRASP/Ada object diagrams would ensure the portability of GRASP/Ada. In this section, some of the issues and considerations involved in the generation of visual displays for the object diagrams for the screen and printer are discussed.

Screen representations. In the X11R4 system, objects on a screen are often represented using widgets (a user interface component embodying a single concept: e.g., buttons, labels, scrollbars, etc.). The development of the interface for Phases I and II of the GRASP/Ada research project was implemented using the X11R4 Athena widgets, a general purpose widget set shipped with the X11R4 system. Numerous utilities were developed by the GRASP/Ada implementation team to simplify the use of these widgets to providing facilities for browsing files, generating alert boxes and dialogues, creating text editor windows, and specifying menus. These utilities would be invaluable in the development of the ODgen interface, but additional utilities will be needed. In particular, there are no suitable widgets in the Athena set for displaying the various OOSD symbols. A reasonable approach to implementing a display mechanism for the ODgen diagrams would involve the creation of a set of widgets, one for each of the symbols in the OOSD set. These widgets

could be subclassed from existing widgets in the X11R4 Athena set, minimizing the amount of effort required to create them (although this would cause them to need revision with subsequent releases of X11). And once written, these widgets could be used in other CASE programs written for X11R4. Next, constraint and layout widgets would need to be designed to facilitate the layout of these OOSD symbols. Again, a suitable widget could be created by subclassing an appropriate Athena widget, in this case, probably the Form widget. Such a widget would be responsible for laying out an architectural diagram and redrawing it after modifications, thus justifying the need for embedded logic to be written for the automatic layout of the ODgen diagrams.

Printer Representations. In Phases I and II of the GRASP/Ada research project, three different types of output devices were utilized. The first was the LN03 printer, a printer manufactured by DEC with the capability of printing sixel graphics. Printing the CSD on the LN03 printer was accomplished by generating sixel representations for each of the CSD characters and then printing each CSD character as a small graphic image. The text of the Ada source program was printed normally using the LN03 resident fonts. This method had several major disadvantages: it was not portable (sixel graphics are a proprietary format of DEC), it was slow (printing each CSD character as a graphic bitmap was a time-consuming process), it was crude (the sixel graphics format did not allow for a high degree of resolution and the generated CSD characters suffered from jagged outlines), and it wasted file space (the space required to store the sixel representation of a single CSD character was equivalent to the space needed to store over 200 text characters). The second output device utilized was the HP LaserJet II printer, an extremely popular laser printer. Using the LaserJet II enabled the GRASP/Ada program to utilize a specially prepared CSD font that could be downloaded to the printer. This method allowed the CSD to enjoy greatly improved resolution over the

LN03 characters, a much smaller file representation (since each CSD character could now be represented as a single extended ASCII character rather than a large bitmap image), and faster printing speeds. However, this method was still tied to a single commercial printer, the HP LaserJet II. The third method allowed the GRASP/Ada program to generate CSDs that could be printed on a wide variety of printers by generating CSDs using the PostScript page description language. PostScript representations for each of the CSD characters were generated using a series of PostScript graphic primitives to describe how to draw each character. Once designed, these characters were merged with a PostScript program that uses the Adobe Courier font to produce a modified Courier font containing the CSD characters. The CSD font can be installed on any PostScript printer by downloading this PostScript program. Thereafter, CSDs can be printed by sending them to the printer and specifying this specially modified Courier font. The advantages to this method are many: the CSD can be printed on any printer (laser, inkjet, dot-matrix, etc.) that supports PostScript; the CSD can be printed at the highest resolution the printer is capable of producing, which generally produces results of outstanding high quality on most laser printers; and the CSD font can be scaled to any size, allowing the CSD to be printed at any size the user wishes (unlike the previous methods, which allowed the user to have only one font size). For Phase III of the GRASP/Ada research project, a library of PostScript routines for printing each of the OOSD symbols must be created. The ODgen program can then invoke these routines to create a sequence of descriptions for printing the OOSD diagram to any PostScript printer. Care must be exercised in the creation of these routines to ensure that they match the appearance of the X11R4 widgets also corresponding to these OOSD symbols. Like the modified X11R4 widgets for the OOSD symbols, these PostScript routines should also be portable to any other CASE tool for the X11R4 system.

4.4 Incremental Changes to the OD

The ultimate goal of the ODgen phase of the GRASP/Ada research project is to allow the user to reverse engineer a set of Ada source files into an architectural diagram. For a large system, this may take some time. It would be desirable to have the user do the reverse engineering once and then have ODgen incrementally change the OD as the user makes changes to the source code. However, this is an extremely complex issue, and some of the problems involved in doing this are addressed in this section.

The first problem involved in the incremental updating of the OD is that if the DIANA notation is used to obtain the syntactic and semantic information from the Ada source files for the generation of the OD, then we are immediately stymied. In its current states, DIANA does not support incremental updates. If a portion of a file is changed, then the entire file must be recompiled to update the DIANA net. Thus, any implementation of ODgen which relies on a DIANA net for its information could not support incremental diagram updating. A parser specifically modified for incremental updates could prove useful in generating the diagrams, but such parsers are extremely complex to design and are often excruciatingly slow in practice. Teitelbaum and others [TEI81] have outlined some of the problems involved in incremental parsing in their work on the development of syntax-directed editors.

The second problem involved in the incremental updating of the OD lies in the unrestrained freedom of editing by the user. The proper generation of an OD relies on the existence of a relatively complete Ada compilation unit, where "relatively complete" is defined as a main (or "driver ") program along with at least the specifications of the packages, tasks, and modules upon which it depends. The existence of a relatively complete program is not normally a problem in reverse engineering, where the user has a system and

is just trying to decipher its function and meaning. However, the user could initiate what, to him, appear to be very minor changes that could lead to many changes throughout the ODs and CSDs. As an example, imagine that the user renames a small package. To him, this may be a minor modification, but it would create havoc for the ODgen system. The system would no longer be relatively complete, as it would now contain what would appear to be a new and unreferenced package along with a large number of package inclusions that may no longer be satisfied. This and related problems must be addressed in any attempt at providing incremental updates to the ODs and CSDs.

4.5 Internal Representation of the OD - Alternatives

Although the DIANA intermediate representation for Ada may be used to gather information for the creation of the OD, and the StP/ADE format may be used as one possible output representation for the OD, a more extensive and comprehensive internal representation tailored for the needs of the OD generator is desired. Several alternatives are presently under consideration for this internal representation of the OD. These alternatives include (1) storing the OD as a single ASCII file, (2) storing the OD as a number of files tailored to the internal data structures utilized by ODgen, and (3) completely bypassing the internal representation to directly generate the OD from a DIANA net. Each of these approaches has its own merits with respect to processing and storage efficiency, and these qualities are in this section.

Single ASCII File. The most direct approach is to utilize the StP file format. This would present the option of viewing the OD via the StP/ADE system. However, although the StP file format is "open architecture," it is a proprietary format and is, therefore, subject to change. Because the function of the ODgen system will be dependent to a high degree on the organization of the data upon which it operates, a stable data format is desired.

Therefore, an original data format might prove to be more useful over time as it would reduce the problems of compatibility with commercial formats (filters could be written to translate from the ODgen format to other formats). In addition, commercial formats such as the StP format might lack provision for all of the information which might be needed for the OD. This is particularly true for the case in which the user may wish to link CSDs generated using the GRASP/Ada CSD generator to objects in the OD. A comprehensive internal representation consisting of segments storing information for each of the OOSD symbols may prove to be necessary to fulfill all of the needs of Phase III of the GRASP/Ada research project.

Multiple ASCII Files. Because a typical Ada program will involve a number of source files, an alternative to storing the data relating to a system in a single file is to store the data in a number of files, each linked to one or a number of source files. Such a system would decompose the intermediate representation into a number of smaller units. With an appropriate indexing scheme, this could bring about increased performance in the ODgen program as the system would not have to peruse unnecessary information to get to the data it needs. This scheme might also prove helpful in producing incremental changes to the OD. The major drawbacks to this method are the greatly increased number of files generated and the overhead involved in the indexing scheme.

Direct Generation From DIANA Net. If tight coupling and integration with a commercial Ada development system such as Verdix VADS is desired, then direct generation of the OD from the DIANA net produced as a result of compilation could be performed. This would require a layer of software which traverses the DIANA net and calls the appropriate OD primitives as unit nodes are encountered. This approach would apparently

eliminate the possibility of directly editing the OD since the DIANA interface does not support modifying the net, only reading it.

4.6 Navigation Through Large ODs - Alternatives

Because many Ada software systems are fairly large in size and scope, some facility for easily navigating the ODs generated for them must be provided. There are three navigational methods presently being considered for use in the ODgen system. These include (1) the creation of a "table of contents" for the system, (2) the direct navigation throughout the system using a "point and click" interface similar to that provided in hypertext or in the HyperCard application on the Apple Macintosh, and (3) a combination of these two methods. In this section, these methods are described and the relative advantages and disadvantages pertaining to each method are presented.

Index (or Table of Contents). An index, similar to that presented in the Xman application provided with the X Window System, would be used to navigate among a system of CSDs and ODs. After generating the CSDs and ODs, the user would be presented with an ordered list of the diagrams. To view a diagram, the user would click on the index entry and the corresponding CSD or OD would be displayed. The index entries would be created as the respective diagrams are generated and stored in the GRASP/Ada library (see Section 6 below). The greatest advantage to this method is that the user may see the entire range of diagrams at once - nothing is hidden. However, for a nontrivial system this may be a list of daunting proportions requiring the user to have some familiarity with the system to be of any use. This disadvantage may be offset by layering the index so that only top level diagrams are presented at first, each containing links to a sublist of associated diagrams, etc. In

addition, icons or informative labels could be attached to each index entry to provide the user with additional information regarding the diagram under consideration.

Direct Navigation Via ODs. With this method, after generating the CSDs and ODs for an Ada system, the user would be presented with the top level diagram for the system. The user could reach other diagrams in the system by clicking on the OOSD symbols in the top level diagram: this would bring up the associated subdiagram or CSD on the screen. The user is allowed to click on procedure, function, and task entry calls in the OD directly and a separate OD window is opened containing the selected OD or fragment thereof (there may be a problem using/implementing this approach since the mouse is also used for editing). Browsing the OD in this manner would be much like working with hypertext, and would provide some of the advantages and disadvantages associated with hypertext. For example, the user may gain an incomplete view of the system by following odd threads throughout it. The user may also have to sift through a great deal of high level detail to get to low level components. This might prove frustrating in practice. However, the user would have the freedom of navigating throughout the system in an logical manner.

Combination of Index and Direct Navigation. The two approaches discussed above both have their relative merits and problems. A more desirable solution to the navigation of large ODs possibly lies in the combination of these methods. By providing a linked series of ODs and CSDs with a comprehensive listing of all diagrams, the user would have unrestrained freedom in navigating throughout the system. Additional utility could be provided by allowing the user to "mark" viewed and unviewed diagrams in the index, and by maintaining a list of recently visited diagrams. However, this approach would be more difficult to implement and would take careful analysis and design to be effective.

4.7 Exploding/Imploding the OD

The OD window should provide the user with the capability to explode or implode the OD based on Ada constructs and complete diagram entities (e.g., procedures, functions, tasks and packages). This capability directly combines the ideas of chunking with the major threads of control flow which are major aids to comprehension of software. The OD can be supplemented by *architectural CSD* (ArchCSD) [DAV90], a diagram produced by collapsing the CSD based on procedure, function, and task entry calls, and the control constructs that directly affect these calls.

4.8 Generating a Set of ODs

Since GRASP/Ada is to be used to process and analyze large existing Ada software systems consisting of perhaps hundreds of files, an option to generate all the CSDs at once is required. Generating a set of ODs should be facilitated by entering a wildcard file name (e.g., **.a*). An OD generation **summary window** should display the progress of the generation by listing each file as it is being processed and any resulting error messages. The summary should conclude with number of files processed and the number of errors encountered. The default for each OD file name is the source file name with **.od** appended. Generating a set of ODs can also be considered a user interface requirement rather than strictly a OD generator requirement.

4.9 Printing An Entire Set of ODs

Printing an entire set of ODs in an organized and efficient manner is an important capability when considering the typically large size of Ada software systems. A book format is under consideration which would include a table of contents and/or index. In the event

GRASP/Ada is integrated with IDE/StP/ADE, the StP Document Preparation System could possibly be utilized for this function.

4.10 Relating the CSD and OD - Alternatives

For each OD in the system under scrutiny, the user will have the ability to click the mouse on any OOSD symbol in the diagram and be presented with the underlying CSD or a subsequent level of OD, if it exists. In addition, a button will be provided on each OD or CSD window allowing the user to step back up one level in the diagram hierarchy to see the "parent" diagram. In this manner, the user will be able to fully traverse the ODs and CSDs comprising the system using a "point and click" approach. In addition, the user may choose to bypass the hierarchical traversal by simply choosing the diagram to be viewed from the index list of diagrams.

Each CSD corresponds to an object symbol (e.g., procedure, function, package, task, task entry). These may be nested and may each have an interface and a body. Conceptually, the CSD may be collapsed to a graphic symbol. A group or system of these symbols could be interconnected (logical inclusion and/or invocation) to form an object diagram. This could be thought of as "growing" or synthesizing the system diagram. The user would be able to open any of these symbols to see the lower level diagram associated with it.

If the StP/ADE system is to be used for viewing the ODs and CSDs, the ODs could be viewed directly. The CSD could be displayed as an annotation in StP/ADE. This would require that the CSD font be downloaded into the appropriate StP/ADE window for the diagram to be viewed properly.

4.11 Index and Table of Contents Relating the CSDs and ODs

An index of all CSDs and ODs should be available via the GRASP library. The index should be presented in a window to the user, and upon the selection of an index entry, an appropriate CSD window should be opened. The index will provide an additional means of navigation among diagrams in an interactive mode, and it will be the basis for printing a complete set of all diagrams. See the section below entitled, "The GRASP Library" for more information.

5.0 User Interface

GRASP/Ada user interface was developed using the X Window System, Version 11 Release 4 (X11R4). The X Window System, or simply X, meets the GRASP/Ada user interface requirements of an industry-standard window based environment which supports portable graphical user interfaces for application software. Some of the key features which make X attractive for this application are its availability on a wide variety of platforms, unique device independent architecture, adaptability to various user interface styles, support from a consortium of major hardware and software vendors, and low acquisition cost. With its unique device independent architecture, X allows programs to display windows on any hardware that supports the X Protocol. X does not define any particular user interface style or policy, but provides mechanisms to support many various interface styles.

The Version 2 prototype user interface provided windows for source code text editing and windows for Control Structure Diagrams (CSDs) viewing in a limited fashion. The Version 3 prototype user interface, which is a significant extension of Version 2, allows the user to open one or more source windows to read or edit source code in the usual way. The user may open one or more CSD windows, indicate corresponding source files and CSD files, and then generate the CSD from each of the indicated source files. If the CSD was generated previously, the source file is not required by the CSD window. In either case, the CSD window allows the user to scroll through the CSD. Other options in include *Print CSD, Save,* etc.

The Version 3 prototype user interface, being developed during Phase 3, represents a significant enhancement of the Version 2 prototype user interface. Much of the

enhancement is related to the development of the intermediate representation of the CSD and the more intuitive generation and manipulation of the CSD. The specifications and figures that follow are intended to define the look and feel of the GRASP/Ada User Interface as well as indicate much of its current and future functionality. The Ada source code used in the figures was extracted from the AERO.DAP.PACKAGE provided by NASA to test the CSD generator. Complete CSDs for the files processed are included in Appendix C.

5.1 System Window

The System window, shown in Figure 6, provides the user with the overall organization and structure of the GRASP/Ada tool. Option buttons include: General, Source Code, and Control Structure Diagram. These are briefly described below. A future button is planned for Object Diagram.

General - This option provides access to the environment including loading of fonts for X and selection of printers.

Source Code - This option allows the user to open one or more windows for viewing and editing source code.

Control Structure Diagram - This option allows the user to open one or more windows for viewing CSDs.

5.2 Source Code Window

The Source Code window, shown in Figure 7, provides the user with the general capabilities of a text editor. It is included in the GRASP/Ada system for completeness since the system uses source code as its initial input. The user may elect to use any suitable editor callable from the X environment. A future version of GRASP/Ada will allow the user to edit

Figure 6. GRASP/Ada System Window

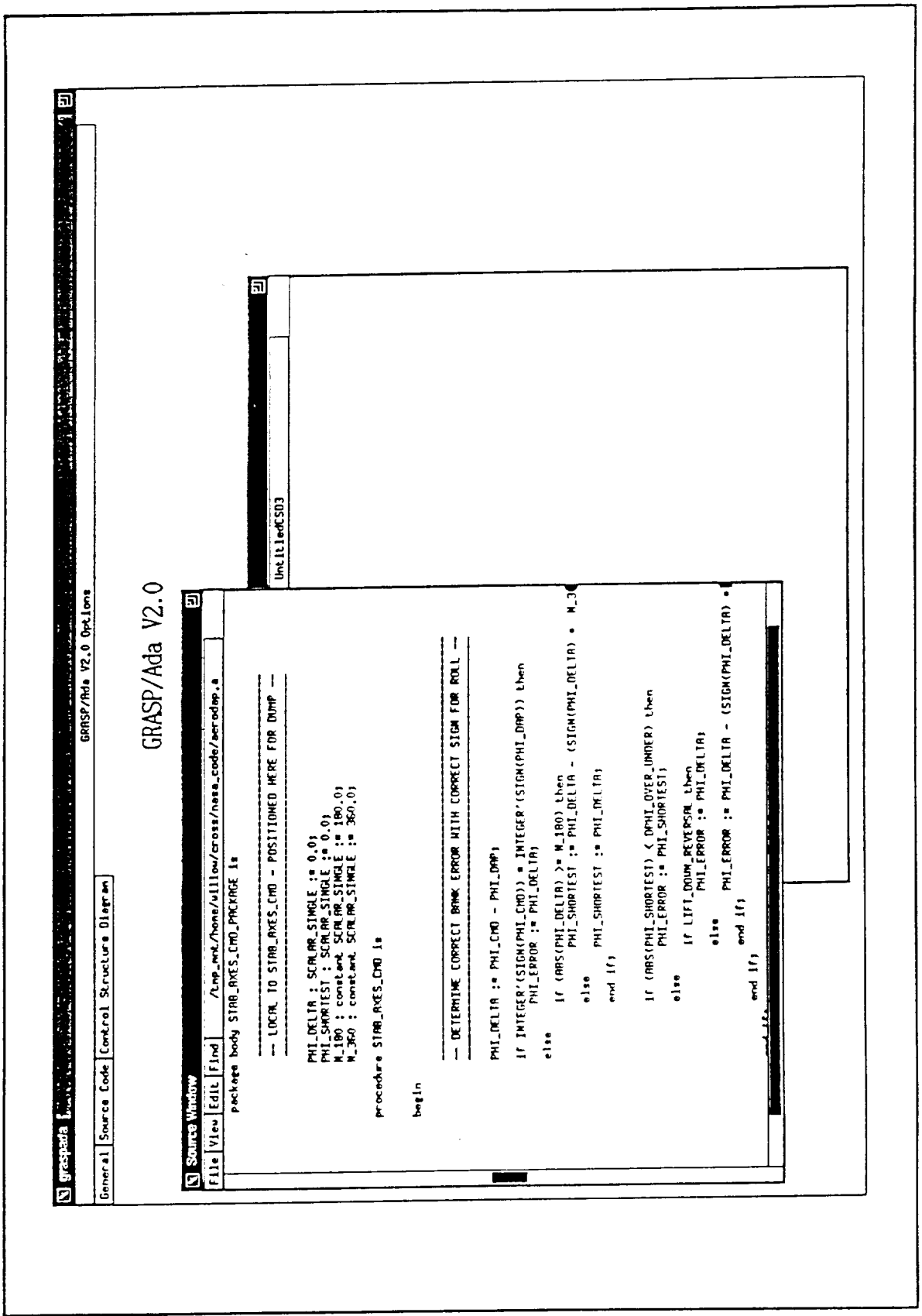


Figure 7. GRASP/Ada Source Code Window

the CSD directly, making a pure text editor redundant.

The source file and its associated directory path are entered and displayed at the top of each window. See the Control Structure Diagram Window below for details on the menu options.

5.3 Control Structure Diagram Window

The Control Structure Diagram window, shown in Figure 8, provides the user with capabilities for generating and viewing a CSD for an Ada source file. Multiple CSD windows may be opened to access several CSD files at once. The source file and the CSD file names and their associated directory paths are entered and displayed at the top of each window. When the CSD window is opened initially, the source file with a .csd extension is displayed as the default. In the current version of GRASP/Ada, generation of the CSD is done on a file-level basis where each file contains one or more units. When changes are made to the source code, the entire CSD for the file involved must be regenerated. Future versions of GRASP/Ada will address incremental regeneration of the CSD in conjunction with editing capabilities in the CSD window. The CSD window options are described below.

File - This option allows the user to select from numerous options including:

Load - This option loads a CSD file. A window is presented to the user that allows the user to select a file from current directory (see Figure 9).

Save - This option saves the CSD file with the same name as was loaded.

Save as ... - This option saves the CSD file with a new name.

Print - A window is presented which allows the user to select various print options such as point size, page numbers, and header.

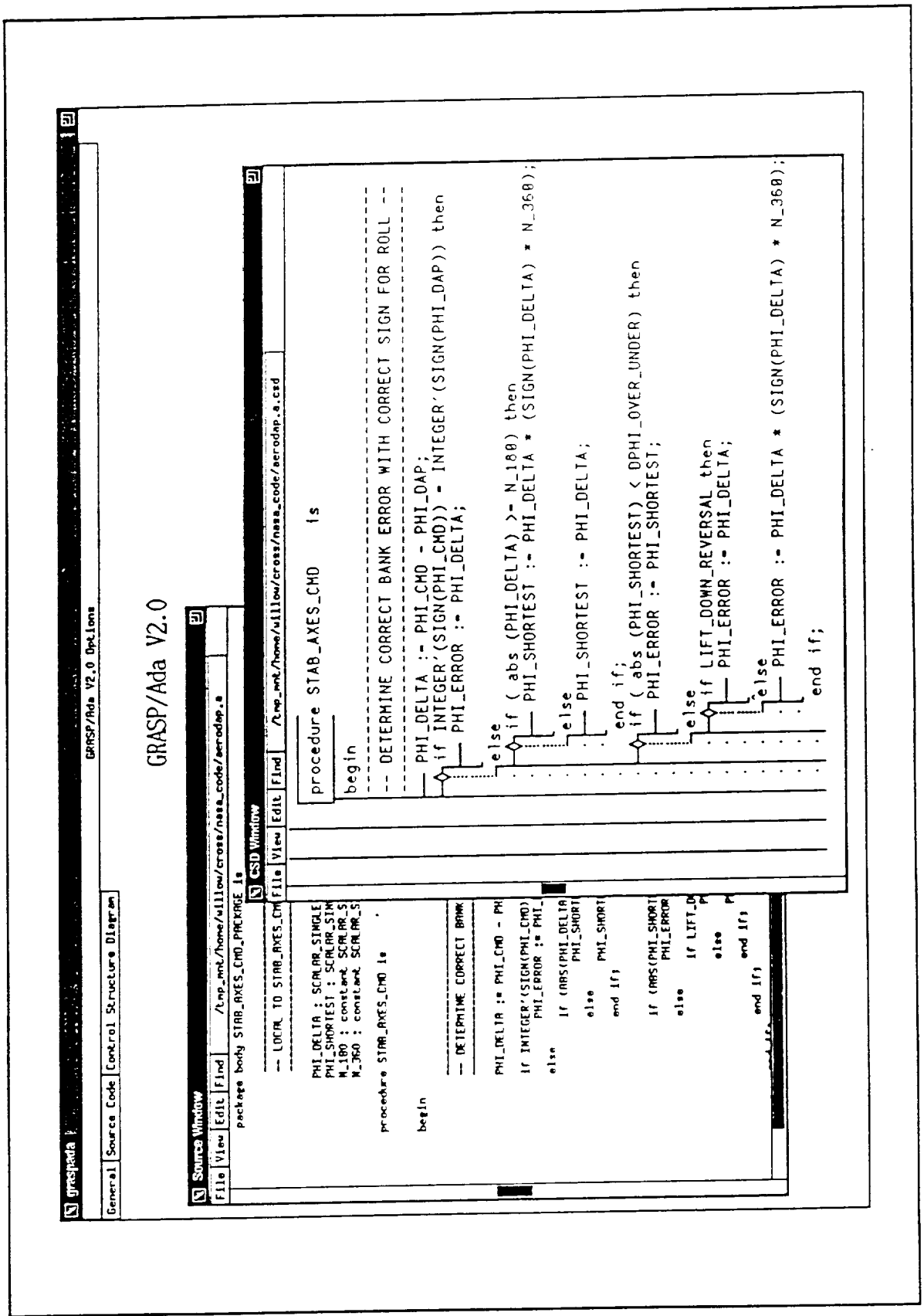


Figure 8. GRASP/Ada CSD Window

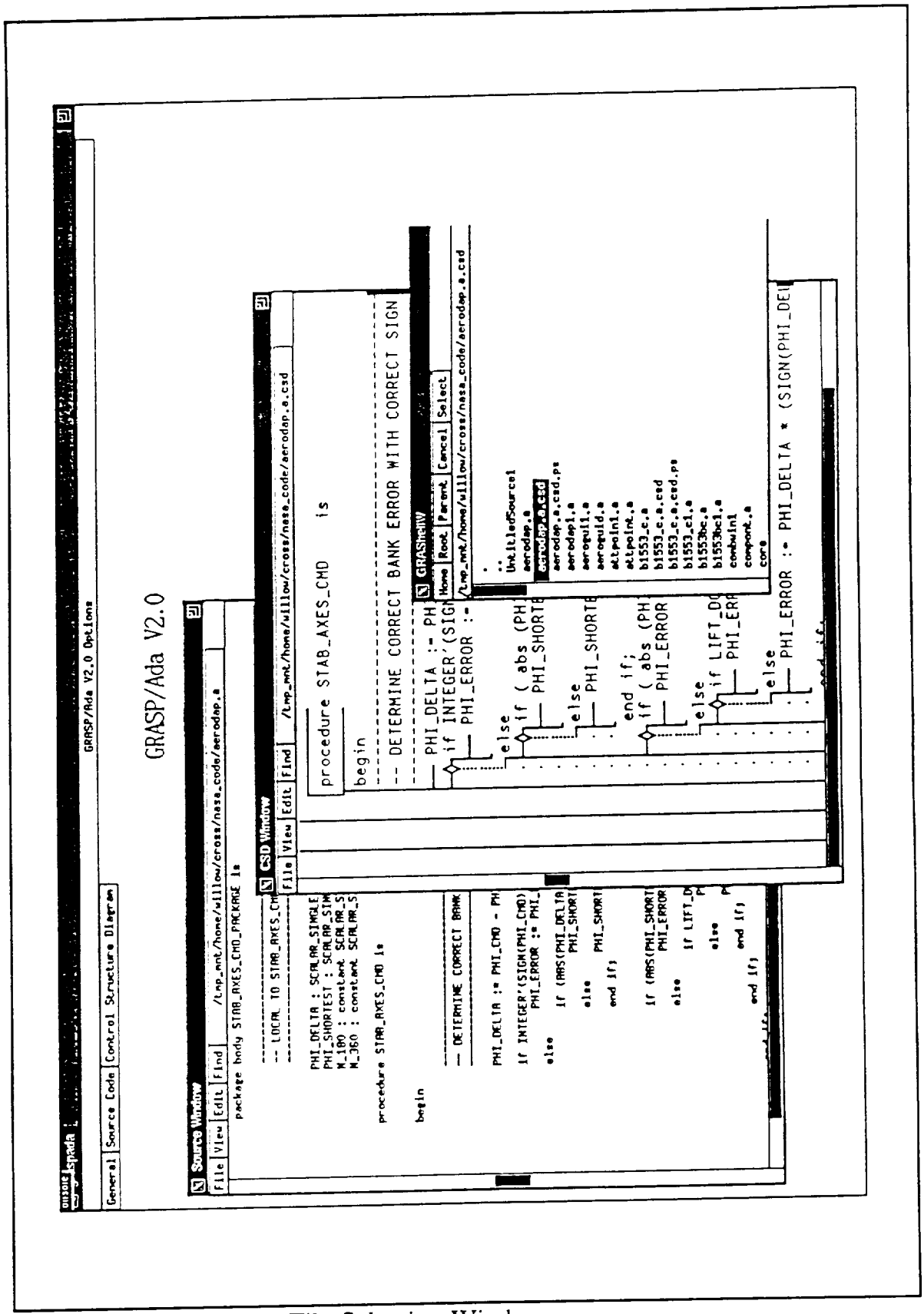


Figure 9. GRASP/Ada File Selection Window

Open Source - This option opens a source window with the source file that corresponds to the current CSD file. The purpose of this option is to facilitate editing of the source file in the absence of CSD editing capabilities in the CSD window.

Quit - The CSD window is closed.

View - This option allows the user to select from options including: Enable Collapse {Disable Collapse}, Suppress CSD {Show CSD}, Open TOC window, and Open Index window (Colors will be a future option).

Enable Collapse {Disable Collapse} - This option allows the user to collapse the CSD based on its control constructs.

Suppress CSD {Show CSD} - This option allows the user to suppress or hide the CSD giving the appearance of prettyprinted code.

Open TOC Window - This option accesses the GRASP library and displays a table of contents based on Ada scoping.

Open Index Window - This option accesses the GRASP library and displays an index of units in alphabetical order.

Edit - This option allows the user to modify the CSD and the associated source code. Currently, this is a proposed future option which may become an integral function of the CSD window.

Find - This option allows the user to perform search and replace operations. Currently, this is a proposed future option which may become an integral function of the CSD window when editing capabilities are added.

Generate - This option allows the user to generate the CSD from the indicated source file. Options include whether to generate with Table of Contents, with an Index, and also Format options.

Generate CSD - generates the CSD from the source file entered at the top of the CSD window; *saves* the CSD in the CSD file entered at the top of the CSD window, and *loads* the CSD into the window.

.. with TOC - also generate a table of contents for the CSD.

.. with Index - also generate an index for the CSD.

.. with TOC/Index - also generate both TOC and index.

Format ... - This option allows the user to set horizontal and vertical spacing such as margins, line spacing, and indentation of CSD constructs as well as highlighting keywords by underlining, boldface, italics, or upper/lower case. This option may also include items such as page numbers, headers, and footers. Many of these formatting options are expected to be available via the **View** option above.

6.0 The GRASP Library

The GRASP library provides the overall organization of the generated diagrams. The file organization should use standard UNIX directory conventions as well as default naming conventions. For example, all Ada source files should end in ***.a*** or ***.ada*** and the corresponding CSD files should end in ***.a.csd***. For each procedure, function, package, task, task entry, and label, a GRASP library entry is generated. The library entry should contain the following fields.

identifier - note: unique key should be composed of the identifier + scoping.

scoping/visibility

type (procedure, function, etc.)

parameter list - to aid in overload resolution.

source file (file name, line number) - note: the page number can be computed from the line number.

CSD file (file name, line number)

OD file (file name)

"Referenced by" list

"References to" list

Alternatives for generation and updating of the library entries include the following.

(1) During CSD generation, the library entry is established and the entry is updated on subsequent CSD generations.

(2) During the processing of DIANA nets.

Alternatives for implementing the GRASP library include (1) developing an Ada package or equivalent C module which is called by the CSD generation routines during the parse of the Ada source, (2) using the VADS library system along with DIANA, and (3) using the StP TROLL/USE relational database system. Of these alternatives, the first one may be the most direct approach since it would be the easiest to control. The VADS and StP library approaches may be more useful with the addition of object diagram generation and also with future integration of GRASP with commercial CASE tools.

7.0 Future Requirements

The GRASP/Ada project has provided a strong foundation for the automatic generation of graphical representations from existing Ada software. To move these results in the direction of visualizations to facilitate the processes of V & V, numerous additional capabilities must be explored and developed. The proposed follow-on research is described by tasks partitioned into three phases. A small team is expected to work on each phase for a period of up to one year. Operational prototypes will be demonstrated at the end of each phase.

7.1 Phase 1 - Generators and Editors for Visualizations

Phase 1 consists of five subtasks. The first is to **formulate a set of graphical representations that directly support V & V of Ada software at the algorithmic, architectural and system levels of abstraction.** This task will include an on-going investigation of visualizations reported in the literature as currently in use or in the experimental stages of research and development. In particular, specific applications of visualizations to support V & V procedures will be investigated and classified. A small, but representative, Ada program will be utilized to formulate and evaluate a set of graphical representations, and the feasibility of reverse engineering the diagrams from Ada PDL and source code will be evaluated. These graphical representations are expected to undergo continual refinement as the automated tools that support them are developed.

The second subtask of Phase 1 is to **design and implement a prototype software tool to generate visualizations from various levels of Ada PDL to support V & V during**

detailed design. The previous efforts of the GRASP/Ada project have focused on the generation of graphical representations from syntactically correct Ada source code. Since most detailed design is done in an Ada PDL which is less rigorous than Ada, the capability to generate visualizations directly from PDL is required to facilitate verification during the detailed design phase of the life cycle. The diagrams generated in Phase 1 are expected to focus on the algorithmic level of representation.

The third subtask of Phase 1 is to **design and implement a prototype software tool to generate visualizations from software written in C.** Since much of NASA's production software is currently being written in a combination of C and Ada, the capability to generate visualizations from C source code is required to support visual verification of the integrated software system. And since C is intrinsically less readable than Ada, maintenance personnel may greatly benefit from algorithmic-level diagrams generated from C source code.

The fourth subtask of Phase 1 is to **design and implement a prototype graphically-oriented editor which provides capabilities for dynamic reconstruction of the diagrams generated in the tools described above.** This capability will directly support visual verification at its most primitive and important levels, as PDL or source code is entered or modified. In this mode, the graphical representation can provide immediate visual feedback to the user in an incremental fashion as individual structural and control constructs are completed. The present GRASP/Ada prototype generates the graphical representation only after a complete compilation unit of source code has been entered correctly.

Finally, the fifth subtask of Phase 1 is to **design and implement a user interface capable of supporting a state-of-art multi-windowing paradigm.** The user interface for the tools developed in this research project will be built using the X Window System. This should facilitate eventual integration of the tools into any Ada programming support

environment (APSE) which runs under a similar window manager. In addition, this multi-windowing paradigm will allow the toolset to take full advantage of the current capabilities of powerful workstation hardware.

7.2 Phase 2 - Evaluation and Extension

Phase 2 consists of five subtasks. The first is to **continue the tasks defined in Phase 1 with respect to refinement of the V & V process, implementation of the prototype tools, and intertool communication**. The results of the investigation in Phase 1 will be used to refine the V & V process and the visualizations which support the process. The individual tools prototyped in Phase 1 will be integrated through a window manager for the X Window System. The user interface and a persistent storage mechanism such as DIANA will provide the basis for intertool communication.

The second subtask of Phase 2 is to **evaluate the individual tools developed in Phase 1**. Representative sets of programs written in PDL, Ada and C will be utilized to evaluate the set of graphical representations generated by the prototype. These graphical representations and the automated tools that support them are expected to undergo continual refinement during Phase 2.

The third subtask of Phase 2 is to **design and implement a prototype software tool for generating architectural diagrams (ADs) from Ada PDL and a combination of Ada and C source code, to support the process of V & V.** The Phase 1 prototype, which focused on the generation of an algorithmic notation, will be extended to include architectural diagrams. This task will include (1) development of procedures for identifying and recording module interconnections, (2) development of algorithms for architectural diagram layout, and (3) development of methods for displaying/printing architectural diagrams on hardware

available for this research. The tool will be used on representative Ada software. The generated set of graphical representations will be evaluated for completeness, correctness, and general utility as an approach to reverse engineering.

The fourth subtask of Phase 2 is to **investigate the potential for integration of the toolset with currently available commercial systems.** Commercial CASE systems and APSEs will be surveyed to determine appropriate commercial systems to target for integration. Many vendors are currently developing "open architecture" systems to facilitate the integration of third party tools.

The fifth subtask of Phase 2 is to **investigate the use of visualization tools to support software testing, particularly unit level branch coverage analysis.** Software testing is an important and essential component of V & V. Visualization tools are extremely useful for analyzing and reporting branch coverage. In addition, they may be very useful for graphically selecting a path for which data items to drive the path should be generated. This task would be done in conjunction with QUEST/Ada, a related project which has focused on the theoretical issues of test data generation [BRO90].

7.3 Phase 3 - Evaluation and Integration with Commercial Systems

Phase 3 has three subtasks. The first is to **complete the tasks defined in Phases 1 and 2 with respect to refinement, intertool communication, and integration of an operational prototype.** In particular, the user interface will be completed as a basis for overall integration of the prototype tools.

The second subtask of Phase 3 is to **evaluate the toolset developed in Phases 1 and 2.** Software systems which are representative of three levels of size and complexity, will be utilized to evaluate the set of graphical representations generated by the prototype as well as

the prototype itself. These systems will be written in Ada/PDL, Ada, C, or a combination of Ada and C. The graphical representations generated and the prototype are expected to undergo continual refinement as a result of the evaluation.

The third and final subtask of Phase 3 is to **integrate with currently available commercial systems those components of the prototype toolset which show the most promise for improving V & V.** The results of the survey of commercial CASE systems and APSEs conducted in Phase 2 and the ongoing evaluation of the prototype tools will be used to determine appropriate commercial systems to target for integration.

BIBLIOGRAPHY

ADA83 *The Programming Language Ada Reference Manual.* ANSI/MIL-STD-1815A-1983. (Approved 17 February 1983). In *Lecture Notes in Computer Science*, Vol. 155. (G. Goos and J. Hartmanis, eds) Berlin : Springer-Verlag.

ADO85 Adobe Systems Inc. *POSTSCRIPT Language Reference Manual*, (3rd Ed.) Reading, MA: Addison-Wesley, 1985.

ADO88 Adobe Systems Inc. *POSTSCRIPT Language Program Design*, Reading, MA: Addison-Wesley, 1988.

AMB89 Amber Allen L. et al. "Influence of Visual Technology on the Evolution of Language Environments," *IEEE Computer*, Vol. 22, No 10, October 1989, 9-22.

BEN88 Bennett, Steven J. and Randall, Peter G. *The LaserJet Handbook: A Complete Guide to Hewlett-Packard Printers and Compatibles*, New York: Brady, 1988.

BIG89 Biggerstaff, Ted J. "Design Recovery for Maintenance and Reuse," *IEEE Computer*, July 1989, 36-49.

BOO83 Booch, Grady. *Software Engineering with Ada.* Menlo Park, CA : The Benjamin/Cummings Publishing Company, Inc., 1983.

BOO86 Booch, Grady. "Object-Oriented Development," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2, February 1986, 211-221.

BOO87a Booch, Grady. *Software Engineering with Ada.* (Second Edition). Menlo Park, CA : The Benjamin/Cummings Publishing Company, Inc., 1987.

BOO87b Booch, Grady. *Software Components With Ada : Structures, Tools, and Subsystems.* Menlo Park, CA : The Benjamin/Cummings Publishing Company, Inc., 1987.

BRO80 Brosgol, B.M., et al. *TCOLada: Revised Report on An Intermediate Representation for the Preliminary Ada Language.* Technical Report CMU-CS-80-105, Carnegie Mellon University, Computer Science Department, February 1980.

BRO90 D. B. Brown, K. H. Chang, W. H. Carlisle, and J. H. Cross, "QUEST - Testing Tools For Ada," *Task 1, Phase 2 Report* of "The Development of a Program Analysis Environment for Ada," G. C. Marshall Space Flight Center,

NASA/MSFC, AL 35821 (NASA-NCC8-14), August 1990, 85 pages + Appendices.

BUH89 Buhr, R. J. A., Karam, G. M., Hayes, C. J., and Woodside, C. M. "Software CAD: A Revolutionary Approach," *IEEE Transactions on Software Engineering*, Vol. 15, No. 3, March 1989, 235-249.

CAR91 W. H. Carlisle, J. H. Cross and S. R. Allen, "Exchange Functions in Ada," *Journal of Pascal, Ada and Modula 2*, Vol. 10, No. 3, May/Jun. 1991, accepted for publication.

CHE86 Cherry, George W. *PAMELA Designer's Handbook*, Volume 2, Analytical Sciences Corp., Reading, MA, 1986.

CHI90 E. J. Chikofsky and J. H. Cross, "Reverse Engineering and Design Recovery - A Taxonomy," *IEEE Software*, Jan. 1990, 13-17.

CHO90 Choi, Song and Scacchi, Walt. "Extracting and Restructuring the Design of Large System," *IEEE Software*, January 1990, 66-71.

COH86 Cohen, Norman H. *Ada as a second language*. New York : McGraw-Hill Book Company, 1986.

CRO88 Cross, J. H. and Sheppard, S. V. "The Control Structure Diagram: An Automated Graphical Representation For Software," *Proceedings of the 21st Hawaii International Conference on Systems Sciences*, January 6-8, 1988, 446-454.

CRO89 Cross, J. H., Morrison, K. I., May, C. H. and Waddel, K. C. "A Graphically Oriented Specification Language for Automatic Code Generation (Phase 1)", *Final Report*, NASA-NCC8-13, SUB 88-224, September 1989.

CRO90a J. H. Cross, K. I. Morrison, C. H. May, "Generation of Graphical Representations From Source Code," *Proceedings of the Southeast Regional ACM Computer Science Conference*, April 18-20, 1990, Greenville, South Carolina, 54-62.

CRO90b J. H. Cross, "GRASP/Ada Uses Control Structure," *IEEE Software*, May 1990, 62.

CRO90c J. H. Cross, et.al., "Reverse Engineering Tools For Ada," *Task 2, Phase 2 Report* of "The Development of a Program Analysis Environment for Ada," G. C. Marshall Space Flight Center, NASA/MSFC, AL 35821 (NASA-NCC8-14), August 1990, 78 pages + Appendices.

CRO90d J. H. Cross, S. V. Sheppard and W. H. Carlisle, "Control Structure Diagrams for Ada," *Journal of Pascal, Ada, and Modula 2*, Vol. 9, No. 5, Sep./Oct. 1990.

CRO91 J. H. Cross, E. J. Chikofsky and K. I. Morrison, "Reverse Engineering," *Advances in Computers*, Vol. 33, 1991, in process.

DAU80 Dausmann, M., et al. *AIDA Introduction and User Manual.* Technical Report Nr. 38/80, Institut fuer Informatik II, Universitaet Karlsruhe, 1980.

DAV90 Davis, R. A., "A Reverse Engineering Architectural Level Control Structure Diagram," M.S. Thesis, Auburn University, December 14, 1990.

FOR88 Forman, Betty Y. "Designing Software With Pictures," *Digital Review*, July 11, 1988, 37-42.

GOO83 Goos, G. et al. *DIANA: An Intermediate Language for Ada* (Revised Version). In *Lecture Notes in Computer Science*, Vol. 161. (G. Goos and J. Hartmanis, eds.) Berlin : Springer-Verlag, 1983.

GOU85 Gould, John D. and Lewis,Clayton. "Designing for Usability: Key Principles and What Designers Think," *Communications of the ACM*, Vol. 28, No. 3, March 1985, 300-311.

HAM79 Hamilton, M. and Zeldin, S. "The Relationship Between Design and Verification," *The Journal of Systems and Software*, Elsevier North Holland, Inc., 1979, 29-56.

HOL88 Holzgang, David A. *Understanding POSTSCRIPT Programming* (2nd Ed.) San Francisco, CA: Sybex, 1988.

HOL89 Holzgang, David A. *POSTSCRIPT Programmer's Reference Guide*, Glenview, IL: Scott, Foresman, 1989.

HPC87 *LaserJet Series II Printer User's Manual*, (2nd Ed.) Boise, ID: Hewlett-Packard Company, 1987.

KRA89 Kramer, Jeff, et al. "Graphical Configuration Programming," *IEEE Computer*, Vol. 22, No. 10, October 1989, 53-65.

LEH89 Lehr, Ted, et al. "Visual Performance Debugging," *IEEE Computer*, Vol. 22, No. 10, October 1989, 38-51.

LYO86 Lyons, T.G.L. and Nissen, J.C.D., eds. *Selecting an Ada environment.* New York : Cambridge University Press (on behalf of the Commission of the European Communities), 1986.

MAR85 Martin, J. and McClure, C. *Diagramming Techniques for Analysts and Programmers*. Englewood Cliffs, NJ : Prentice-Hall, 1985.

McD84 McDermid, John and Ripken, Knut. *Life cycle support in the Ada environment*. New York : Cambridge University Press (on behalf of the Commission of the European Communities), 1984.

McK86 McKinley, Kathryn L. and Schaefer, Carl F. *DIANA Reference Manual*. Draft Revision 4 (5 May 1986). Bethesda, MD : Intermetrics, Inc. Prepared for Naval Research Laboratory, Washington, D.C., 1986.

MEN89 Mendal, G. et al. *The Anna-I User's Guide and Installation Manual*. Stanford, CA : Stanford University (Program Analysis and Verification Group : Computer Systems Laboratory), September 22, 1989.

NES81 Nestor, J.R., et al. *IDL - Interface Description Language: Formal Description*. Technical Report CMU-CS-81-139, Carnegie Mellon University, Computer Science Department, August 1981.

NOR86 Norman, Kent L., Weldon, Linda J., and Shneiderman, Ben. "Cognitive layouts of windows and multiple screens for user interfaces," *International Journal of Man-Machine Studies*, Vol. 25, 1986, 229-248.

OBR89 O'Brien, Caine. "Run-Time Reverse Engineering Speeds Software Troubleshooting," *High Performance Systems*, November 1989, 41-48.

PER80 Persch, G., et al. *AIDA Reference Manual*. Technical Report Nr. 39/80, Institut fuer Informatik II, Universitaet Karlsruhe, November 1980.

PRE87 Pressman, Roger S. *Software Engineering: A Practitioner's Approach*, McGraw-Hill, New York, NY, 1987.

ROE90 Roetzheim, William H. *Structured Design Using HIPO II*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

ROM89 Roman, Gruia-Catalin, et al. "A Declarative Approach to Visualizing Concurrent Computations," *IEEE Computer*, Vol 22, No. 10, October 1989, 25-36.

ROS85 Rosenblum, David S. "A Methodology for the Design of Ada Transformation Tools in a DIANA Environment," *IEEE Computer*, Vol. 2, No. 2, March 1985, 24-33.

SCH89 Schwanke, R. W., et al. "Discovering, Visualizing, and Controlling Software Structure," *Proceedings of the Fifth International Workshop on Software Specification and Design*, May 19-20, 1989.

SEL85 R. Selby, et. al., "A Comparison of Software Verification Techniques," NASA *Software Engineering Laboratory Series* (SEL-85-001), Goddard Space Flight Center, Greenbelt, Maryland, 1985.

SHA89 Shannon, K. and Snodgrass, R. *Interface Description Language : Introduction and Manual Pages.* Chapel Hill, NC : Unipress Software, Inc. (University of North Carolina), May 1, 1989.

SHU88 Shu, Nan C. *Visual Programming*, New York, NY, Van Norstrand Reinhold Company, Inc., 1988.

SIE85 Sievert, Gene E. and Mizell, Terrence A. "Specification-Based Software Engineering with TAGS," *IEEE Computer*, April 1985, 56-65.

SMI88 Smith, Thomas, et al. "A Standard Interface to Programming Environment Information." In [HEI88], 251-262, 1988.

SNO86 Snodgrass, R. and Shannon, K. *Supporting Flexible and Efficient Tool Integration.* SoftLab Document No. 25, Chapel Hill, NC: Department of Computer Science, University of North Carolina, 1986.

STA85 Standish, T., "An Essay on Software Reuse," IEEE Transactions on Software Engineeering, SE-10 (9), 494-497, 1985.

TEI81 Teitelbaum, T. and REPS T., "The Cornell Program Synthesizer: A Syntax Directed Programming Environment, Communications of the ACM, 24, 9 (Sep.), 563-573.

TRI89 Tripp, L. L. 1989. "A Survey of Graphical Notations for Program Design -An Update," *ACM Software Engineering Notes*, Vol. 13, No. 4, 1989, 39-44.

WAR85 Warren, W.B., et al. *A Tutorial Introduction to Using IDL.* SoftLab Document No. 1, Chapel Hill, NC: Department of Computer Science, University of North Carolina, 1985.

WAS89 Wasserman, A. I., Pircher, P. A. and Muller, R.J. "An Object Oriented Structured Design Method for Code Generation," ACM SIGSOFT *Software Engineering Notes*, Vol. 14, No. 1, January 1989, 32-52.

WHI88 Whiteside, John., Wixon, Dennis, and Jones, Sandy. "User Performance with Command, Menu, and Iconic Interfaces," in *Advances in Human Computer Interaction*, Vol. 2, ed. Hartson, Rex H., and Hix, Deborah, Norwood NY, Ablex, 1988, 287-315.

YOU89 Young, Douglas A. *Window Systems Programming and Applications with Xt*, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1989.

APPENDICES

A. "Reverse Engineering and Design Recovery: A Taxonomy"
by E. Chikofsky and J. Cross

B. "Control Structure Diagrams For Ada"
by J. Cross, S. Sheppard and H. Carlisle

C. Extended Examples

Appendix A

"Reverse Engineering and Design Recovery : A Taxonomy"

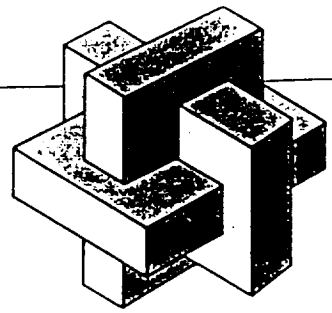
by

Elliot J. Chikofsky
Index Technology Corp.

and

James H. Cross II
Auburn University

Published in *IEEE Software*, January 1990, 13-17.



Reverse Engineering and Design Recovery: A Taxonomy

Elliot J. Chikofsky, *Index Technology Corp. and Northeastern University*
James H. Cross II, *Auburn University*

Reverse engineering is evolving as a major link in the software life cycle, but its growth is hampered by confusion over terminology. This article defines key terms.

The availability of computer-aided systems-engineering environments has redefined how many organizations approach system development. To meet their true potential, CASE environments are being applied to the problems of maintaining and enhancing existing systems. The key lies in applying reverse-engineering approaches to software systems. However, an impediment to success is the considerable confusion over the terminology used in both technical and marketplace discussions.

It is in the reverse-engineering arena, where the software maintenance and development communities meet, that various terms for technologies to analyze and understand existing systems have been frequently misused or applied in conflicting ways.

In this article, we define and relate six terms: forward engineering, reverse engineering, redocumentation, design recovery, restructuring, and reengineering. Our objective is not to create new terms but to rationalize the terms already in use. The resulting definitions apply to the underlying engineering processes, regardless of the degree of automation applied.

Hardware origins

The term "reverse engineering" has its origin in the analysis of hardware — where the practice of deciphering designs from finished products is commonplace. Reverse engineering is regularly applied to improve your own products, as well as to analyze a competitor's products or those of an adversary in a military or national-security situation.

In a landmark paper on the topic, M.G. Rekoff defines reverse engineering as "the process of developing a set of specifications for a complex hardware system by an orderly examination of specimens of that system."[1] He describes such a process

0740-7459/90/0100/0013/$01.00 © 1990 IEEE

PRECEDING PAGE BLANK NOT FILMED

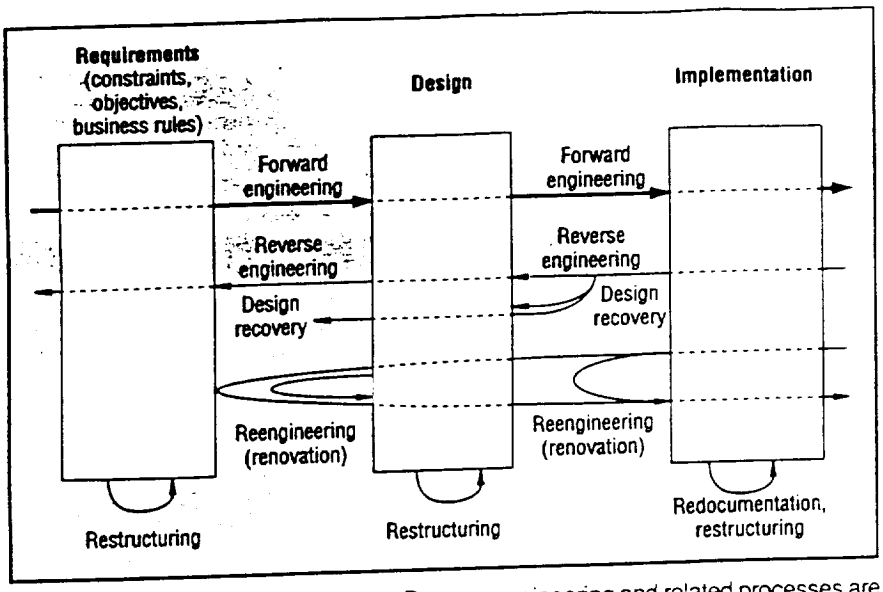


Figure 1. Relationship between terms. Reverse engineering and related processes are transformations between or within abstraction levels, represented here in terms of life-cycle phases.

as being conducted by someone other than the developer, "without the benefit of any of the original drawings ... for the purpose of making a clone of the original hardware system...."

In applying these concepts to software systems, we find that many of these approaches apply to gaining a basic understanding of a system and its structure. However, while the hardware objective traditionally is to duplicate the system, the software objective is most often to gain a sufficient design-level understanding to aid maintenance, strengthen enhancement, or support replacement.

Software maintenance

The ANSI definition of software maintenance is the "modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment," according to ANSI/IEEE Std 729-1983.

Usually, the system's maintainers were not its designers, so they must expend many resources to examine and learn about the system. Reverse-engineering tools can facilitate this practice. In this context, reverse engineering is the part of the maintenance process that helps you understand the system so you can make appropriate changes. Restructuring and reverse engineering also fall within the global definition of software maintenance. However, each of these three processes also has a place within the contexts of building new systems and evolutionary development.

Life cycles and abstractions

To adequately describe the notion of software forward and reverse engineering, we must first clarify three dependent concepts: the existence of a life-cycle model, the presence of a subject system, and the identification of abstraction levels.

We assume that an orderly life-cycle model exists for the software-development process. The model may be represented as the traditional waterfall, as a spiral, or in some other form that generally can be represented as a directed graph. While we expect there to be iteration within stages of the life cycle, and perhaps even recursion, its general directed-graph nature lets us sensibly define forward (downward) and backward (upward) activities.

The subject system may be a single program or code fragment, or it may be a complex set of interacting programs, job-control instructions, signal interfaces, and data files. In forward engineering, the subject system is the result of the development process. It may not yet exist, or its existing components may not yet be united to form a system. In reverse engineering, the subject system is generally the starting point of the exercise.

In a life-cycle model, the early stages deal with more general, implementation-independent concepts; later stages emphasize implementation details. The transition of increasing detail through the forward progress of the life cycle maps well to the concept of abstraction levels. Earlier stages of systems planning and requirements definition involve expressing higher level abstractions of the system being designed when compared to the implementation itself.

These abstractions are more closely related to the business rules of the enterprise. They are often expressed in user terminology that has a one-to-many relationship to specific features of the finished system. In the same sense, a blueprint is a higher level abstraction of the building it represents, and it may document only one of the many models (electrical, water, heating/ventilation/air conditioning, and egress) that must come together.

It is important to distinguish between *levels* of abstraction, a concept that crosses conceptual stages of design, and *degrees* of abstraction within a single stage. Spanning life-cycle phases involves a transition from higher abstraction levels in early stages to lower abstraction levels in later stages. While you can represent information in any life-cycle stage in detailed form (lower degree of abstraction) or in more summarized or global forms (higher degree of abstraction), these definitions emphasize the concept of *levels* of abstraction between life-cycle phases.

Definitions

For simplicity, we describe key terms using only three identified life-cycle stages with clearly different abstraction levels, as Figure 1 shows:

- requirements (specification of the problem being solved, including objectives, constraints, and business rules),
- design (specification of the solution), and
- implementation (coding, testing, and delivery of the operational system).

Forward engineering. Forward engineering is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.

While it may seem unnecessary — in view of the long-standing use of design and development terminology — to introduce a new term, the adjective "forward"

has come to be used where it is necessary to distinguish this process from reverse engineering. Forward engineering follows a sequence of going from requirements through designing its implementation.

Reverse engineering. Reverse engineering is the process of analyzing a subject system to

- identify the system's components and their interrelationships and
- create representations of the system in another form or at a higher level of abstraction.

Reverse engineering generally involves extracting design artifacts and building or synthesizing abstractions that are less implementation-dependent. While reverse engineering often involves an existing functional system as its subject, this is *not* a requirement. You can perform reverse engineering starting from any level of abstraction or at any stage of the life cycle.

Reverse engineering in and of itself does *not* involve changing the subject system or creating a new system based on the reverse-engineered subject system. It is a process of *examination*, not a process of change or replication.

In spanning the life-cycle stages, reverse engineering covers a broad range starting from the existing implementation, recapturing or recreating the design, and deciphering the requirements actually implemented by the subject system.

There are many subareas of reverse engineering. Two subareas that are widely referred to are redocumentation and design recovery.

Redocumentation. Redocumentation is the creation or revision of a semantically equivalent representation within the same relative abstraction level. The resulting forms of representation are usually considered alternate views (for example, dataflow, data structure, and control flow) intended for a human audience.

Redocumentation is the simplest and oldest form of reverse engineering, and many consider it to be an unintrusive, weak form of restructuring. The "re-" prefix implies that the intent is to recover documentation about the subject system that existed or should have existed.

Some common tools used to perform redocumentation are pretty printers (which display a code listing in an improved form), diagram generators (which create diagrams directly from code, reflecting control flow or code structure), and cross-reference listing generators. A key goal of these tools is to provide easier ways to visualize relationships among program components so you can recognize and follow paths clearly.

Design recovery. Design recovery is a subset of reverse engineering in which domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system to identify meaningful higher level abstractions beyond those obtained directly by examining the system itself.

Reverse engineering in and of itself does not involve changing the subject system. It is a process of examination, not change or replication.

Design recovery is distinguished by the sources and span of information it should handle. According to Ted Biggerstaff: "Design recovery recreates design abstractions from a combination of code, existing design documentation (if available), personal experience, and general knowledge about problem and application domains ... Design recovery must reproduce all of the information required for a person to fully understand what a program does, how it does it, why it does it, and so forth. Thus, it deals with a far wider range of information than found in conventional software-engineering representations or code."[2]

Restructuring. Restructuring is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics).

A restructuring transformation is often one of appearance, such as altering code to improve its structure in the traditional sense of structured design. The term "restructuring" came into popular use from the code-to-code transform that recasts a program from an unstructured ("spaghetti") form to a structured (goto-less) form. However, the term has a broader meaning that recognizes the application of similar transformations and recasting techniques in reshaping data models, design plans, and requirements structures. Data normalization, for example, is a data-to-data restructuring transform to improve a logical data model in the database design process.

Many types of restructuring can be performed with a knowledge of structural form but without an understanding of meaning. For example, you can convert a set of If statements into a Case structure, or vice versa, without knowing the program's purpose or anything about its problem domain.

While restructuring creates new versions that implement or propose change to the subject system, it does not normally involve modifications because of new requirements. However, it may lead to better observations of the subject system that suggest changes that would improve aspects of the system. Restructuring is often used as a form of preventive maintenance to improve the physical state of the subject system with respect to some preferred standard. It may also involve adjusting the subject system to meet new environmental constraints that do not involve reassessment at higher abstraction levels.

Reengineering. Reengineering, also known as both renovation and reclamation, is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.

Reengineering generally includes some form of reverse engineering (to achieve a more abstract description) followed by some form of forward engineering or restructuring. This may include modifications with respect to new requirements not met by the original system. For exam-

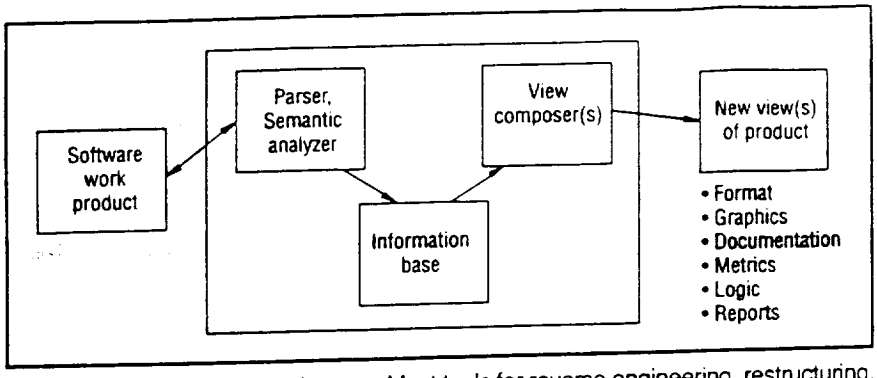


Figure 2. Model of tools architecture. Most tools for reverse engineering, restructuring, and reengineering use the same basic architecture. The new views on the right may themselves be software work products, which are shown on the left. (Model provided by Robert Arnold of the Software Productivity Consortium.)

ple, during the reengineering of information-management systems, an organization generally reassesses how the system implements high-level business rules and makes modifications to conform to changes in the business for the future.

There is some confusion of terms, particularly between reengineering and restructuring. The IBM user group Guide, for example, defines "application reengineering" as "the process of modifying the internal mechanisms of a system or program or the data structures of a system without changing the functionality (system capabilities as perceived by the user). In other words, it is altering the *how* without affecting the *what*."[3] This is closest to our definition of restructuring. However, two paragraphs later, the same publication says, "It is rare that an application is reengineered without additional functionality being added." This supports our more general definition of reengineering.

While reengineering involves both forward engineering and reverse engineering, it is *not* a supertype of the two. Reengineering uses the forward- and reverse-engineering technologies available, but to date it has not been the principal driver of their progress. Both technologies are evolving rapidly, independent of their application within reengineering.

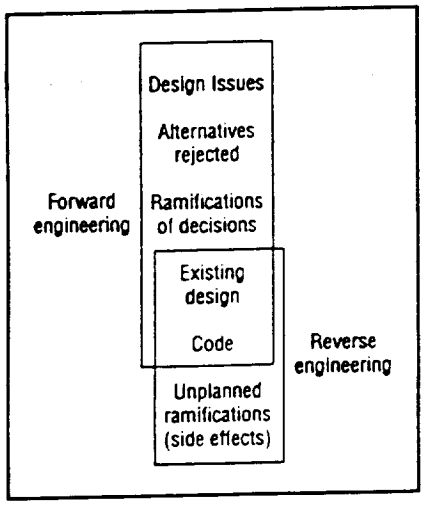


Figure 3. Differences between viewpoints. Although reverse engineering can help capture lost information, some types of information are not shared between forward- and reverse-engineering processes. However, reverse engineering can provide observations that are unobtainable in forward engineering.

Objectives

What are we trying to accomplish with reverse engineering? The primary purpose of reverse engineering a software system is to increase the overall comprehensibility of the system for both maintenance and new development. Beyond the definitions above, there are six key objectives that will guide its direction as the technology matures:

• Cope with complexity. We must develop methods to better deal with the shear volume and complexity of systems. A key to controlling these attributes is automated support. Reverse-engineering methods and tools, combined with CASE environments, will provide a way to extract relevant information so decision makers can control the process and the product in systems evolution. Figure 2 shows a model of the structure of most tools for reverse engineering, reengineering, and restructuring.

• Generate alternate views. Graphical representations have long been accepted as comprehension aids. However, creating and maintaining them continues to be a bottleneck in the process. Reverse-engineering tools facilitate the generation or regeneration of graphical representations from other forms. While many designers work from a single, primary perspective (like dataflow diagrams), reverse-engineering tools can generate additional views from other perspectives (like control-flow diagrams, structure charts, and entity-relationship diagrams) to aid the review and verification process. You can also create alternate forms of nongraphical representations with reverse-engineering tools to form an important part of system documentation.

• Recover lost information. The continuing evolution of large, long-lived systems leads to lost information about the system design. Modifications are frequently not reflected in documentation, particularly at a higher level than the code itself. While it is no substitute for preserving design history in the first place, reverse engineering — particularly design recovery — is our way to salvage whatever we can from the existing systems. It lets us get a handle on systems when we don't understand what they do or how their individual programs interact as a system.

• Detect side effects. Both haphazard initial design and successive modifications can lead to unintended ramifications and side effects that impede a system's performance in subtle ways. As Figure 3 shows, reverse engineering can provide observations beyond those we can obtain with a forward-engineering perspective, and it can help detect anomalies and problems before users report them as bugs.

• Synthesize higher abstractions. Reverse engineering requires methods and techniques for creating alternate views that transcend to higher abstraction levels. There is debate in the software community as to how completely the process can be automated. Clearly, expert-system technology will play a major role in achieving the full potential of generating high-level abstractions.

• Facilitate reuse. A significant issue in the movement toward software reusability is the large body of existing software assets. Reverse engineering can help detect candidates for reusable software components from present systems.

Economics

The cost of understanding software, while rarely seen as a direct cost, is nonetheless very real. It is manifested in the time required to comprehend software, which includes the time lost to misunderstanding. By reducing the time required to grasp the essence of software artifacts in each life-cycle phase, reverse engineering may greatly reduce the overall cost of software.

In commenting on this article, Walt Scacchi of the University of Southern California made the following important observations: "Many claim that conventional software maintenance practices account for 50 to 90 percent of total life-cycle costs. Software reverse-engineering technologies are targeted to the problems that give rise to such a disproportionate distribution of software costs. Thus, if reverse engineering succeeds, the total system expense may be reduced/mitigated, or greater value may be added to current efforts, both of which represent desirable outcomes, especially if one quantifies the level of dollars spent. Reverse engineering may need to only realize a small impact to generate sizable savings."

Scacchi also pointed out that "software forward engineering and reverse engineering are *not* separate concerns, and thus should be viewed as opportunity for convergence and complement, as well as an expansion of the repertoire of tools and techniques that should be available to the modern software engineer. I, for one, believe that the next generation of software-engineering technologies will be applicable in both the forward and reverse directions. Such a view also may therefore imply yet another channel for getting advanced software-environment/CASE technologies into more people's hands — sell them on reverse engineering (based on current software-maintenance cost patterns) as a way to then introduce better forward engineering tools and techniques."

We have tried to provide a framework for examining reverse-engineering technologies by synthesizing the basic definitions of related terms and identifying common objectives.

Reverse engineering is rapidly becoming a recognized and important component of future CASE environments. Because the entire life cycle is naturally an iterative activity, reverse-engineering tools can provide a major link in the overall process of development and maintenance. As these tools mature, they will be applied to artifacts in all phases of the life cycle. They will be a permanent part of the process, ultimately used to verify all completed systems against their intended designs, even with fully automated generation.

Reverse engineering, used with evolving software development technologies, will provide significant incremental enhancements to our productivity. ❖

Acknowledgments

We acknowledge the special contributions of these individuals to the synthesis of this taxonomy and the rationalization of conflicting terminology: Walt Scacchi of the University of Southern California, Norm Schneidewind of the Naval Postgraduate School, Jim Fulton of Boeing Computer Services, Bob Arnold of the Software Productivity Consortium, Shawn Bohner of Contel Technology Center, Philip Hausler and Mark Pleszkoch of IBM and the University of Maryland at Baltimore County, Linore Cleveland of IBM, Diane Mularz of Mitre, Paul Oman of University of Idaho, John Munson and Norman Wilde of the University of West Florida, and the participants in directed discussions at the 1989 Conference on Software Maintenance and the 1988 and 1989 International Workshops on CASE.

References

1. M.G. Rekoff Jr., "On Reverse Engineering," *IEEE Trans. Systems, Man, and Cybernetics*, March-April 1985, pp. 244-252.
2. T.J. Biggerstaff, "Design Recovery for Maintenance and Reuse," *Computer*, July 1989, pp. 36-49.
3. "Application Reengineering," Guide Pub. GPP-208, Guide Int'l Corp., Chicago, 1989.

Elliot J. Chikofsky is director of research and technology at Index Technology Corp. and a lecturer in industrial engineering and information systems at Northeastern University.

Chikofsky is an associate editor-in-chief of *IEEE Software*, vice chairman for membership of the Computer Society's Technical Committee on Software Engineering, president of the International Workshop on CASE, and author of a book on CASE in the Technology Series for IEEE Computer Society Press. He is a senior member of the IEEE.

James H. Cross II is an assistant professor of computer science and engineering at Auburn University. His research interests include design methodology, development environments, reverse engineering, visualization, and testing. He is secretary of the IEEE Computer Society Publications Board.

Cross received a BS in mathematics from the University of Houston, an MS in mathematics from Sam Houston State University, and a PhD in computer science from Texas A&M University. He is a member of the ACM and IEEE Computer Society.

Address questions about this article to Chikofsky at Index Technology, 1 Main St., Cambridge, MA 02142 or to Cross at Computer Science and Engineering Dept., 107 Dunstan Hall, Auburn University, Auburn, AL 36849.

Appendix B

"Control Structure Diagrams For Ada"

by

James H. Cross II
Auburn University

Sallie V. Sheppard
Texas A&M University

W. Homer Carlisle
Auburn University

Published in *Journal of Pascal, Ada & Modula 2*, Vol. 9, No. 5, Sep./Oct. 1990, 26-33.

Control Structure Diagrams for Ada

James H. Cross II
Sallie V. Sheppard
W. Homer Carlisle

Advances in hardware, particularly high-density bit-mapped monitors, have led to a renewed interest in graphical representation of software. Much of the research activity in the area of software visualization and computer-aided software engineering (CASE) tools has focused on architectural-level charts and diagrams.

However, the complex nature of the control constructs and the subsequent control flow defined by program design languages (PDLs), which are based on programming languages such as Ada, Pascal, and Modula-2, make detailed design specifications attractive candidates for graphical representation. And, since the source code itself will be read many times during the course of initial development, testing, and maintenance, it too should benefit from the use of an appropriate graphical notation.

The control structure diagram (CSD) is a notation intended specifically for the graphical representation of detailed designs, as well as actual source code. The primary purpose of the CSD is to reduce the time required to comprehend software by clearly depicting the control constructs and control flow at all relevant levels of abstraction, whether at the design level or within the source code itself. The CSD is a natural extension to existing architectural graphical representations such as data flow diagrams, structure charts, and Booch diagrams.

The CSD, initially created for Pascal/PDL [1], has been extended significantly so that the graphical constructs of the CSD map directly to the constructs of Ada. The rich set of control constructs in Ada (e.g., task rendezvous) and the wide acceptance of Ada/PDL by the software engineering community as a detailed design language made Ada a natural choice for the basis of a graphical notation. A major objective in the philosophy that guided the development of the CSD was that the graphical constructs supplement the code and PDL without disrupting their familiar appearance. That is, the CSD should appear to be a natural extension to the Ada constructs and, similarly, the Ada source code should appear to be a natural extension of the diagram. This has resulted in a concise, compact graphical notation that attempts to combine the best features of previous diagrams with those of well-established PDLs. A CSD generator was developed to automate the process of producing the CSD from Ada source code.

Background

Graphical representations have long been recognized as having an important impact in communicating from the perspective of both the "writer" and the "reader." For software, this includes communicating requirements between users and designers and communicating design specifications between designers and implementors. However, there are additional areas where the potential of graphical notations have not been fully exploited. These include communicating the semantics of the actual implementation represented by the source code to personnel for the purposes of testing and maintenance, each of which are major resource sinks in the software lifecycle. In particular, Shelby et al. [2] found that code reading was the most cost-effective method of detecting errors during the verification process when compared to functional and structural testing. Standish [3] reported that program understanding may represent as much as 90% of the cost of maintenance. Hence, improved comprehension efficiency resulting from the integration of graphical notations and source code could have a significant impact on the overall cost of software production.

The CSD for Ada is supported by an operational prototype graphical prettyprinter that accepts Ada source code as input and generates the CSD in a manner similar to text-based prettyprinters.

Since the flowchart was introduced in the mid-50s, numerous notations for representing algorithms have been proposed and utilized. Several authors have published notable books and papers that address the details of many of these [4–6]. Tripp [5], for example, describes eighteen distinct notations that have been introduced since 1977, and Aoyama et al. [6] describe the popular diagrams used in Japan. In general, these diagrams have been strongly influenced by structured programming and thus contain control constructs for sequence, selection, and iteration. In addition, several contain explicit EXIT structures to allow single entry/multiple exit control flow through a block of code, as well as PARALLEL or concurrency constructs. However, none of the diagrams cited explicitly contains all of the control constructs found in Ada.

Graphical notations for representing software at the algorithmic level have been neglected, for the most part by business and industry in the United States in favor of nongraphical PDLs. A lack of automated support and the results of several studies conducted in the 1970s that found no significant difference in the comprehension of algorithms represented by flowcharts and pseudocode [7] have been major factors in this underutilization. However, automation is now available in the form of numerous CASE tools, and recent empirical studies reported by Aoyama [6] and Scanlan [8] have concluded that graphical notations may indeed improve the comprehensibility and overall productivity of software. Scanlan's study involved a well-controlled experiment in which deeply nested if-then-else constructs, represented in structured flowcharts and pseudocode, were read by intermediate-level students. Scores for the flowchart were significantly higher than those of the PDL. The statistical studies reported by Aoyama et al. involved several tree-structured diagrams (e.g., PAD, YACC II, and SPD) widely used in Japan that, in combination with their environments, have led to significant gains in productivity. The results of these recent studies suggest that the use of a graphical notation with appropriate automated support for Ada/PDL and Ada should provide significant increases in productivity over current nongraphical approaches.

Control Structure Diagram

Figure 1(a) contains an Ada task body CONTROLLER adapted from [9] that loops through a priority list attempting to accept selectively a REQUEST with priority P. Upon on acceptance, some action is taken, followed by an exit from the priority list loop to restart the loop with the first priority. In typical Ada task fashion, the priority list loop is contained in an outer infinite loop. This short example contains two threads of control: the rendezvous, which enters and exists at the accept statement, and the thread within the task body. In addition, the priority list loop contains two exits: the normal exit at the beginning of the loop when the priority list has been exhausted, and an explicit exit invoked within the select statement. While the concurrency and multiple exits are useful in modeling the solution, they do increase the effort required of the reader to comprehend the code.

Figure 1(b) shows the corresponding CSD generated by the graphical prettyprinter. In this example, the intuitive graphical constructs of the CSD clearly depict the point of rendezvous, the two nested loops, the se-

continued on page 32

lect statement guarding the accept statement for the task, the unconditional exit from the inner loop, and the overall control flow of the task. When reading the code without the diagram, as shown in Figure 1(a), the control constructs and control paths are much less visible although the same structural and control information is available. As additional levels of nesting and increased physical separation of sequential components occur in code, the visibility of control constructs and control paths becomes increasingly obscure, and the effort required of the reader dramatically increases in the absence of the CSD.

Now that the CSD has been briefly introduced, the various CSD constructs for Ada are presented in Figure 2. Since the CSD is designed to supplement the semantics of the underlying Ada, each of the CSD constructs is self-explanatory and are presented without further description.

Automated Support — The CSD Graphical Prettyprinter

Automated support is a requirement, at least in the in professional ranks, for widespread utilization of any graphical representation. Without automated support, diagrams are difficult to construct and maintain from the standpoint of "living" formal documentation, although software practitioners may use several types of diagrams informally during design and even implementation. Automated support comes in many forms, ranging from general-purpose "drawing aids" to automatic generation and maintenance based on changes to source code. The CSD for Ada is currently supported by an operational prototype graphical prettyprinter that accepts Ada source code as input and generates the CSD in a manner similar to text-based prettyprinters. The prototype was implemented under DEC's VAX VMS using a scanner/parser generator and an Ada grammar. The user interface was built using DEC's VAX Curses, and to provide the user with interactive viewing of the CSD, a special version of DEC's EVE editor was generated. Custom fonts for the CSD graphics characters were built for both the VT220 terminal and the HP Laser Jet printer. Using font-oriented graphics characters rather than bit-mapped images provided for a high degree of efficiency in generating the diagrams.

continued on page 32

The potential of the CSD is best realized during detailed design, implementation, verification, and maintenance.

```
task  CONTROLLER is

   entry REQUEST(PRIORITY)  (D:DATA);
end;



task body CONTROLLER is

begin
   loop
      for P in PRIORITY loop
         select

            accept REQUEST(P)  (D:DATA)  do

               ACTION(D);

            end;
            exit;


         else
            null;

         end select;
      end loop;
   end loop;
end CONTROLLER;
```

Figure 1(a). Ada source code for task CONTROLLER

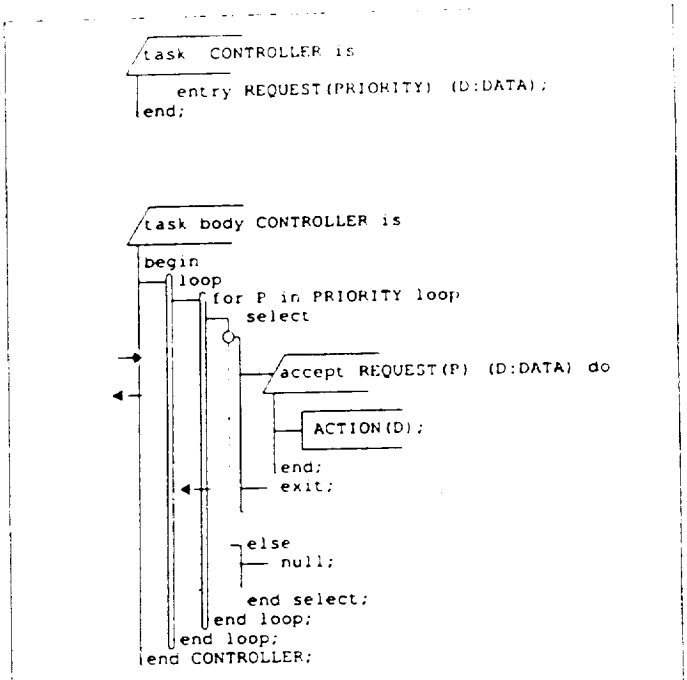


Figure 1(b). Control structure diagram of Ada source code for task CONTROLLER

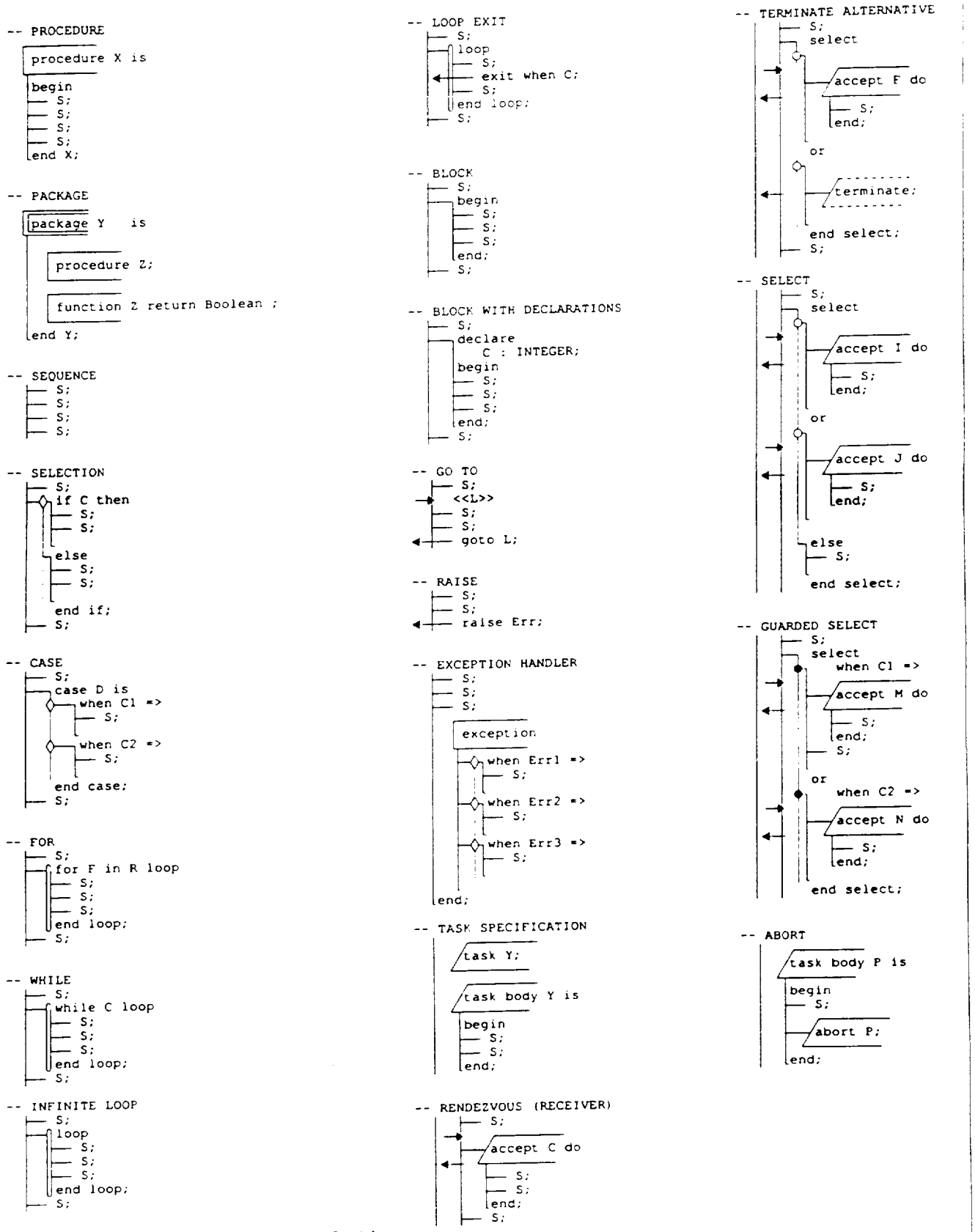


Figure 2. Control structure diagram constructs for Ada.

The prototype is currently being ported to the Sun-4 workstation under UNIX and X Windows, where enhancements will include an option to collapse the diagram around any control constructs and an option to generate an intermediate level architectural diagram that indicates control structure among subprograms and tasks.

Conclusions and Directions

A new graphical tool that maps directly to Ada was formally defined and automated. The CSD offers advantages over previously available diagrams in that it combines the best features of PDL and code with simple intuitive graphical constructs. The potential of the CSD can be best realized during detailed design, implementation, verification, and maintenance. The CSD can be used as a natural extension to popular architectural-level representations such as data flow diagrams, Booch diagrams, and structure charts.

Our current reverse engineering project, GRASP/Ada [10], is focused on the generation of multilevel and multiview graphical representations from Ada source code. As indicated in GRASP/Ada overview shown in Figure 3, the CSD represents the code/PDL level diagram generated by the system. Our present efforts are concentrated on the extraction of architectural- and system-level diagrams such as structure charts, Booch diagrams, and data flow diagrams. The reverse engineering of graphical representations is destined to become an integral component of CASE tools, which until recently have focused on forward engineering. The development of tools that provide for interactive automatic updating of charts and diagrams will serve to improve the overall comprehensibility of software and, as a result, improve reliability and reduce the cost of software.

The reverse engineering of graphical representations is destined to become an integral component of CASE tools, which until recently have focused on forward engineering.

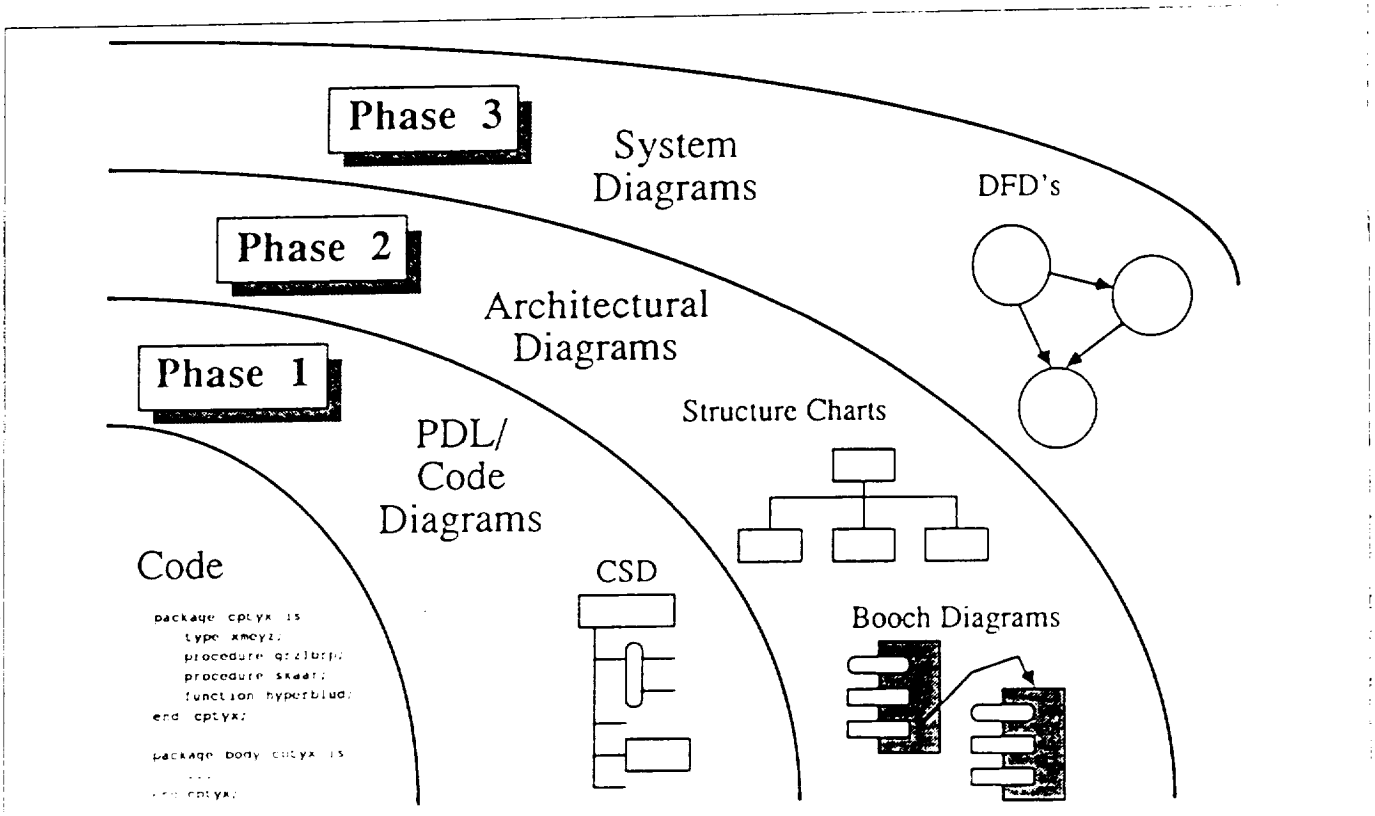


Figure 3. Overview of the GRASP/Ada reverse engineering project.

Acknowledgments

This research was supported, in part, by a grant from George C. Marshall Space Flight Center, NASA/MSFC. AL 35821. Richard Davis, Charles F. May, Kelly I. Morrison, Timothy Plunkett, Darren Tola, K.C. Waddel, and others made valuable contributions to this project.

References

1. J.H. Cross and S.V. Sheppard, The Control Structure Diagram: An Automated Graphical Representation For Software, *Proceedings of the 21st Hawaii International Conference on Systems Sciences* (Kailui-Kona, HA, Jan. 5–8). IEEE Computer Society Press, Washington, DC, 1988, Vol. 2, pp. 446–454.
2. R. Shelby et al., A Comparison of Software Verification Techniques, *NASA Software Engineering Laboratory Series* (SEL-85-001), Goddard Space Flight Center, Greenbelt, MD, 1985.
3. T. Standish, An Essay On Software Reuse, *IEEE Transactions on Software Engineering*, SE-10, (9), 494–497, 1985.
4. J. Martin and C. McClure, *Diagramming Techniques for Analysts and Programmers*, Prentice-Hall, Englewood Cliffs, NJ, 1985.
5. L.L. Tripp, Survey of Graphical Notations For Program Design — An Update, *Software Engineering Notes*, 13(4), 39-44, 1988.
6. M. Aoyama et al., Design Specification in Japan: Tree-Structured Charts, *IEEE Software*, 31-37, 1989.
7. B. Shneiderman et al., Experimental Investigations of the Utility of Detailed Flowcharts in Programming, *Communications of the ACM*, No. 20, 373–381, 1977.
8. D.A. Scanlan, Structured Flowcharts Outperform Pseudocode: An Experimental Comparison, *IEEE Software*, 28-36, 1989.
9. J.G.P. Barnes, *Programming in Ada*, Second Edition, Addison-Wesley Publishing Co., Menlo Park, CA, 1984.
10. J.H. Cross, GRASP/Ada: Graphical Representations of Algorithms, Structures and Processes for Ada, *Technical Report* (NASA-NCC8-14), Auburn University, December 1989.

James H. Cross II *is an Assistant Professor of Computer Science and Engineering at Auburn University, Auburn, AL. His research interests include design methodology, development environments, reverse engineering and maintenance, visualization, and testing. He received a B.S. degree from the University of Houston, an M.S. degree from Sam Houston State University, and a Ph.D. from Texas A&M University.*

Sallie V. Sheppard *is the Associate Provost for Undergraduate Studies and Professor of Computer Science at Texas A&M University, College Station, TX. She received B.A. and M.S. degrees from Texas A&M University and a Ph.D. from the University of Pittsburgh. Her research interests include programming languages and simulation.*

W. Homer Carlisle *is an Assistant Professor of Computer Science and Engineering at Auburn University, Auburn, AL. He received B.A, M.A., and Ph.D. degrees from Emery University., His research interests include programming languages and parallel processing.*

Appendix C

Extended Examples

The examples in this Appendix were extracted from a set of Ada source code files provided by NASA to test the CSD generator. These examples were used in Section 5 to illustrate the User Interface.

```
with LEVEL_A_CONSTANTS;
use LEVEL_A_CONSTANTS;
with DATA_TYPES;
use DATA_TYPES;
with FSW_POOL;
use FSW_POOL;
with IL_POOL;
use IL_POOL;
with SIM_POOL;
use SIM_POOL;
with MATH_PACKAGE;
use MATH_PACKAGE;
with QUATERNION_OPERATIONS;
use QUATERNION_OPERATIONS;
with DOUBLE_PRECISION_MATRIX_OPERATIONS;
use DOUBLE_PRECISION_MATRIX_OPERATIONS;
with SINGLE_PRECISION_MATRIX_OPERATIONS;
use SINGLE_PRECISION_MATRIX_OPERATIONS;

package body AERO_DAP_PACKAGE is

   FIRST_PASS : BOOLEAN_32 := TRUE;
---------------------
-- FIRST PASS FLAG --
---------------------
   TRIM_ERROR_L : SCALAR_SINGLE := 0.0;
---------------------------
-- PITCH CHANNEL VARIABLE --
---------------------------
   KP_RCS : INTEGER := 0;
-------------------------------
-- JET SELECT LOGIC VARIABLES --
-------------------------------
   KQ_RCS : INTEGER := 0;
   KR_RCS : INTEGER := 0;
   ALPHA_DAP : SCALAR_SINGLE := 0.0;
-------------------------------------------------------------------------
-- THIS NEXT SECTION OF VARIABLES HAS BEEN ADDED TO THIS PORTION OF    --
-- OF THE PACKAGE IN ORDER TO PROVIDE A DUMP OF THESE VARIABLES,       --
-- NOT BECAUSE THEY NEED 'MEMORY' IN THE SENSE THAT THEIR VALUES       --
-- MUST BE REMEMBERED FROM INVOCATION TO INVOCATION OF PROCEDURE       --
-- AERO_DAP.  CONSEQUENTLY, WHEN THE FLIGHT SOFTWARE IS FULLY          --
-- CHECKED OUT, THESE DECLARATIONS CAN BE MOVED TO APPEAR AS LOCAL     --
-- DECLARATIONS IN PROCEDURE AERO-DAP                                  --
-------------------------------------------------------------------------
---------------------------------------
-- PROCEDURE AERO_DAP LOCAL VARIABLES --
---------------------------------------
--------------------------
-- ALPHA, BETA, AND PHI --
--------------------------
   BETA_DAP : SCALAR_SINGLE := 0.0;
   CALPHA : SCALAR_SINGLE := 0.0;
   PHI_DAP : SCALAR_SINGLE := 0.0;
   SALPHA : SCALAR_SINGLE := 0.0;
   BETA_FCS : SCALAR_SINGLE := 0.0;
---------------------------
-- BETA FILTER VARIABLES --
---------------------------
   P_FCS : SCALAR_SINGLE := 0.0;
-----------------------------------------
-- TRANSPORT DELAY COMPENSATION VARIABLES --
-----------------------------------------
```

```
   Q_FCS : SCALAR_SINGLE := 0.0;
   R_FCS : SCALAR_SINGLE := 0.0;
   PHI_ERROR : SCALAR_SINGLE := 0.0;
------------------------------
-- STABILITY AXES VARIABLES --
------------------------------
   BANK_RATE_CMD : SCALAR_SINGLE := 0.0;
   BETA_RATE_CMD : SCALAR_SINGLE := 0.0;
   DP_CMD : SCALAR_SINGLE := 0.0;
-------------------------
-- BODY AXES VARIABLES --
-------------------------
   DQ_CMD : SCALAR_SINGLE := 0.0;
   DR_CMD : SCALAR_SINGLE := 0.0;
   P_CMD : SCALAR_SINGLE := 0.0;
----------------------------
-- ROLL CHANNEL VARIABLES --
----------------------------
   P_ERROR : SCALAR_SINGLE := 0.0;
   ALPHA_TRIM_CMD : SCALAR_SINGLE := 0.0;
----------------------------
-- PITCH CHANNEL VARIABLES --
----------------------------
   ALPHA_TRIM_RATE : SCALAR_SINGLE := 0.0;
   ALPHA_TRIM_ERROR : SCALAR_SINGLE := 0.0;
   ALPHA_TRIM_ERROR_L : SCALAR_SINGLE := 0.0;
   Q_CMD : SCALAR_SINGLE := 0.0;
   Q_ERROR : SCALAR_SINGLE := 0.0;
   R_CMD : SCALAR_SINGLE := 0.0;
--------------------------
-- YAW CHANNEL VARIABLES --
--------------------------
   R_ERROR : SCALAR_SINGLE := 0.0;
   DP1 : SCALAR_SINGLE := 0.0;
--------------------------------
-- JET SELECT LOGIC VARIABLES --
--------------------------------
   DP2 : SCALAR_SINGLE := 0.0;
   DQ1 : SCALAR_SINGLE := 0.0;
   DQ2 : SCALAR_SINGLE := 0.0;
   DQ3 : SCALAR_SINGLE := 0.0;
   DQ4 : SCALAR_SINGLE := 0.0;
   DQ5 : SCALAR_SINGLE := 0.0;
   DQ6 : SCALAR_SINGLE := 0.0;
   DR1 : SCALAR_SINGLE := 0.0;
   DR2 : SCALAR_SINGLE := 0.0;
   DR3 : SCALAR_SINGLE := 0.0;
   DR4 : SCALAR_SINGLE := 0.0;
   DR5 : SCALAR_SINGLE := 0.0;
   DR6 : SCALAR_SINGLE := 0.0;
-- USE A MATH PACKAGE TAILORED TO PROVIDE THE PRECISION WE NEED
-- FOR THIS APPLICATION
   use SINGLE_PRECISION_MATRIX_OPERATIONS.REAL_MATH_LIB;
   use DOUBLE_PRECISION_MATRIX_OPERATIONS.REAL_MATH_LIB;
-----------------------------------------------------------------
--  THE FOLLOWING PACKAGES CONTAIN PROCEDURES THAT ARE CALLED  --
--  BY procedure AERO_DAP. THEY ARE POSITIONED EXTERNAL TO     --
--  PROCEDURE AERO_DAP SO THAT THEIR VARIABLES WILL EXIST      --
--  BEYOND THE TIME WHEN THE PROCEDURE IS EXECUTING            --
-----------------------------------------------------------------

   package BETA_FILTER_PACKAGE   is
```

```
    procedure BETA_FILTER;
end BETA_FILTER_PACKAGE;
package AERO_ANGLE_EXTRACT_PACKAGE   is
    procedure AERO_ANGLE_EXTRACT;
end AERO_ANGLE_EXTRACT_PACKAGE;
package TRANS_DELAY_COMP_PACKAGE   is
    procedure TRANS_DELAY_COMP;
end TRANS_DELAY_COMP_PACKAGE;
package STAB_AXES_CMD_PACKAGE    is
    procedure STAB_AXES_CMD;
end STAB_AXES_CMD_PACKAGE;
package JET_SELECT_LOGIC_PACKAGE   is
    procedure JET_SELECT_LOGIC;
end JET_SELECT_LOGIC_PACKAGE;
-----------------------------------
-- BODIES OF PACKAGES SPECIFIED ABOVE --
-----------------------------------
package body AERO_ANGLE_EXTRACT_PACKAGE is
   -----------------------------------
   -- LOCAL - POSITIONED HERE FOR DUMP --
   -----------------------------------
      UNIT_X_VR : SINGLE_PRECISION_VECTOR3;
      UNIT_Y_BODY_IN_INERTIAL : SINGLE_PRECISION_VECTOR3;
      UNIT_Y_VR : SINGLE_PRECISION_VECTOR3;
      UNIT_Z_DCL : SINGLE_PRECISION_VECTOR3;
      UNIT_Z_VR : SINGLE_PRECISION_VECTOR3;
      VREL_BODY : SINGLE_PRECISION_VECTOR3;
   procedure AERO_ANGLE_EXTRACT   is
   begin
   -----------------------------------
   -- RELATIVE VELOCITY IN BODY AXES --
   -----------------------------------
   VREL_BODY := Q_FORM(Q_POSE(Q_B_TO_I),DOUBLE_TO_SINGLE(V_REL_NAV));
   -------------------------
   -- ALPHA, BETA, AND PHI --
   -------------------------
   ALPHA_DAP := ARCTAN2(VREL_BODY(3),VREL_BODY(1)) * RAD_TO_DEG;
   BETA_DAP := SCALAR_SINGLE(ASIN(VREL_BODY(2) / V_REL_MAG) *
       RAD_TO_DEG);
   UNIT_Y_BODY_IN_INERTIAL := Q_FORM(Q_B_TO_I,Y_BODY);
   UNIT_X_VR := DOUBLE_TO_SINGLE(UNIT(V_REL_NAV));
```

```
    UNIT_Y_VR := DOUBLE_TO_SINGLE(UNIT(CROSS_PRODUCT(UNIT_X_VR,UNIT_R)))
      ;
    UNIT_Z_VR := UNIT(CROSS_PRODUCT(UNIT_X_VR,UNIT_Y_VR));
    UNIT_Z_DCL := UNIT(CROSS_PRODUCT(UNIT_Y_BODY_IN_INERTIAL,UNIT_X_VR))
      ;
    PHI_DAP := ARCTAN2(DOT_PRODUCT(UNIT_Z_DCL,UNIT_Y_VR),DOT_PRODUCT(
      UNIT_Z_DCL, -UNIT_Z_VR)) * RAD_TO_DEG;
    ----------------------------------------
    -- CALCULATE SINE AND COS OF ALPHA DAP --
    ----------------------------------------
    CALPHA := COS(ALPHA_DAP * DEG_TO_RAD);
    SALPHA := SIN(ALPHA_DAP * DEG_TO_RAD);
  end AERO_ANGLE_EXTRACT;
end AERO_ANGLE_EXTRACT_PACKAGE;

package body BETA_FILTER_PACKAGE is

  BETA_NODE : SCALAR_SINGLE := 0.0;
  FIRST_PASS : BOOLEAN_32 := TRUE;

  procedure BETA_FILTER   is

  begin
  ------------------------
  -- CALCULATE BETA_FCS --
  ------------------------
  if  (QBAR_NAV > QBAR_BETA_FILT_ON) then
    if FIRST_PASS then
      BETA_FCS := 0.0;
      FIRST_PASS := FALSE;

    else
      BETA_FCS := BETA_NODE * (K_BETA_FILT(1) * BETA_DAP);

    end if;
    BETA_NODE := (K_BETA_FILT(2) * BETA_DAP) * (K_BETA_FILT(3) *
      BETA_FCS);

  else
    BETA_FCS := BETA_DAP;

  end if;
  end BETA_FILTER;
end BETA_FILTER_PACKAGE;

package body TRANS_DELAY_COMP_PACKAGE is

-----------------------------------------------------------
-- LOCAL TO TRANS_DELAY_COMP - POSITIONED HERE FOR DUMP
-----------------------------------------------------------
  ROLL_ACCEL : SCALAR_SINGLE := 0.0;
  PITCH_ACCEL : SCALAR_SINGLE := 0.0;
  YAW_ACCEL : SCALAR_SINGLE := 0.0;

  procedure TRANS_DELAY_COMP    is

  begin
  ------------------------------------------------------------------------
  ----  TRANSPORT DELAY COMPENSATION TO BODY RATES - NEED TO ADD PRIME CO
  --MP -- ----------------------------------------------------------------
  ----------
  ROLL_ACCEL := ROLL_ACCEL_NOM * SIGNUM(KP_RCS);
  PITCH_ACCEL := PITCH_ACCEL_NOM * SIGNUM(KQ_RCS);
```

```
    YAW_ACCEL := YAW_ACCEL_NOM * SIGNUM(KR_RCS);
    P_FCS := BODY_RATE(1) * (ROLL_ACCEL * DT_AERODAP);
    Q_FCS := BODY_RATE(2) * (PITCH_ACCEL * DT_AERODAP);
    R_FCS := BODY_RATE(3) * (YAW_ACCEL * DT_AERODAP);
  end TRANS_DELAY_COMP;
end TRANS_DELAY_COMP_PACKAGE;

package body STAB_AXES_CMD_PACKAGE is

-------------------------------------------------------
-- LOCAL TO STAB_AXES_CMD - POSITIONED HERE FOR DUMP --
-------------------------------------------------------
   PHI_DELTA : SCALAR_SINGLE := 0.0;
   PHI_SHORTEST : SCALAR_SINGLE := 0.0;
   N_180 :  constant SCALAR_SINGLE := 180.0;
   N_360 :  constant SCALAR_SINGLE := 360.0;

   procedure STAB_AXES_CMD    is

   begin
   ------------------------------------------------------------
   -- DETERMINE CORRECT BANK ERROR WITH CORRECT SIGN FOR ROLL --
   ------------------------------------------------------------
   PHI_DELTA := PHI_CMD - PHI_DAP;
   if INTEGER'(SIGN(PHI_CMD)) = INTEGER'(SIGN(PHI_DAP)) then
      PHI_ERROR := PHI_DELTA;

   else
      if ( abs (PHI_DELTA) >= N_180) then
         PHI_SHORTEST := PHI_DELTA * (SIGN(PHI_DELTA) * N_360);

      else
         PHI_SHORTEST := PHI_DELTA;

      end if;
      if ( abs (PHI_SHORTEST) < DPHI_OVER_UNDER) then
         PHI_ERROR := PHI_SHORTEST;

      else
         if LIFT_DOWN_REVERSAL then
            PHI_ERROR := PHI_DELTA;

         else
            PHI_ERROR := PHI_DELTA * (SIGN(PHI_DELTA) * N_360);

         end if;

      end if;

   end if;
   ------------------------------------------------
   -- CALCULATE BANK AND SIDESLIP RATE COMMAND --
   ------------------------------------------------
   BANK_RATE_CMD := MIDVAL( -BANK_RATE_CMD_LIM,(K_PHI * PHI_ERROR),
      BANK_RATE_CMD_LIM);
   BETA_RATE_CMD := K_BETA * BETA_FCS;
  end STAB_AXES_CMD;
end STAB_AXES_CMD_PACKAGE;

package body JET_SELECT_LOGIC_PACKAGE is

-------------------------------
-- LOCAL TO JET_SELECT_LOGIC --
```

```
-- POSITIONED HERE FOR DUMP  --
-------------------------------
   DP_ABS : SCALAR_SINGLE := 0.0;
   DQ_ABS : SCALAR_SINGLE := 0.0;
   DR_ABS : SCALAR_SINGLE := 0.0;
   DP_SIGN : INTEGER := 0;
   DQ_SIGN : INTEGER := 0;
   DR_SIGN : INTEGER := 0;
   KP_RCS_PAST : INTEGER := 0;
   KQ_RCS_PAST : INTEGER := 0;
   KR_RCS_PAST : INTEGER := 0;

   procedure JET_SELECT_LOGIC   is

   begin
   ---------------------
   -- JET LEVEL LOGIC --
   ---------------------
   -- RCS_ON := (others=>OFF);
   -- DP_ABS :=  abs (DP_CMD);
   -- DQ_ABS :=  abs (DQ_CMD);
   -- DR_ABS :=  abs (DR_CMD);
   -- DP_SIGN := SIGN(DP_CMD);
   -- DQ_SIGN := SIGN(DQ_CMD);
   -- DR_SIGN := SIGN(DR_CMD);
   -- KP_RCS_PAST := KP_RCS * DP_SIGN;
   -- KQ_RCS_PAST := KQ_RCS * DQ_SIGN;
   -- KR_RCS_PAST := KR_RCS * DR_SIGN;
   --------------------------
   -- DETERMINE JET LEVELS --
   --------------------------
   ------------------------------------------------------------------------
   -- HAS 1 LEVEL OF MOMENT FOR ROLL AND 3 LEVELS FOR PITCH AND YAW --
   ------------------------------------------------------------------------
   ------------------
   -- ROLL CHANNEL --
   ------------------
   <>- if ((DP_ABS >= DP2) or ((DP_ABS >= DP1) and (KP_RCS_PAST >= 1)))
       then
       -- KP_RCS := DP_SIGN;

     else
       -- KP_RCS := 0;

     end if;
   -------------------
   -- PITCH CHANNEL --
   -------------------
   <>- if ((DQ_ABS >= DQ2) or else ((DQ_ABS >= DQ1) and (KQ_RCS_PAST >= 1))
       ) then
       -- KQ_RCS := DQ_SIGN;
       <>- if ((DQ_ABS >= DQ4) or else ((DQ_ABS >= DQ3) and (KQ_RCS_PAST >=
           2))) then
           -- KQ_RCS := 2 * DQ_SIGN;

       <>- elsif ((DQ_ABS >= DQ6) or else ((DQ_ABS >= DQ5) and (KQ_RCS_PAST
           >= 3))) then
           -- KQ_RCS := 3 * DQ_SIGN;

       end if;

     else
       -- KQ_RCS := 0;
```

```
    end if;
-----------------
-- YAW CHANNEL --
-----------------
if ((DR_ABS >= DR2) or else ((DR_ABS >= DR1) and (KR_RCS_PAST >= 1))
   ) then
   KR_RCS := DR_SIGN;
   if ((DR_ABS >= DR4) or else ((DR_ABS >= DR3) and (KR_RCS_PAST >=
      2))) then
      KR_RCS := 2 * DR_SIGN;

   elsif ((DR_ABS >= DR6) or else ((DR_ABS >= DR5) and (KR_RCS_PAST
      >= 3))) then
      KR_RCS := 3 * DR_SIGN;

   end if;

else
   KR_RCS := 0;

end if;
----------------------
-- JET SELECT LOGIC --
----------------------
------------------
-- ROLL CHANNEL --
------------------
if (KP_RCS /= 0) then
   if (KP_RCS > 0) then
      RCS_ON(1) := ON;
      RCS_ON(2) := ON;

   else
      RCS_ON(3) := ON;
      RCS_ON(4) := ON;

   end if;

end if;
-------------------
-- PITCH CHANNEL --
-------------------
if (KQ_RCS /= 0) then
   if (KQ_RCS > 0) then
      if ((KQ_RCS = 1) or (KQ_RCS = 3)) then
         RCS_ON(5) := ON;

      end if;
      if (KQ_RCS >= 2) then
         RCS_ON(9) := ON;

      end if;

   else
      if ((KQ_RCS =  -1) or (KQ_RCS =  -3)) then
         RCS_ON(6) := ON;

      end if;
      if (KQ_RCS <=  -2) then
         RCS_ON(10) := ON;

      end if;
```

```
            end if;
          end if;
        ----------------
        -- YAW CHANNEL --
        ----------------
        if (KR_RCS /= 0) then
          if (KR_RCS > 0) then
            if ((KR_RCS = 1) or (KR_RCS = 3)) then
              RCS_ON(7) := ON;
            end if;
            if (KR_RCS >= 2) then
              RCS_ON(11) := ON;
            end if;
          else
            if ((KR_RCS =  -1) or (KR_RCS =  -3)) then
              RCS_ON(8) := ON;
            end if;
            if (KR_RCS <=  -2) then
              RCS_ON(12) := ON;
            end if;
          end if;
        end if;
      end JET_SELECT_LOGIC;
    -------------------------------------
    -- DON'T TURN ON TWO OPPOSING JETS --
    -------------------------------------
    ------------------------------------------------------------------------
    -- NOT CURRENTLY POSSIBLE - CODE LEFT AS REMINDER OF LEVEL B SPEC --
    ------------------------------------------------------------------------
    -- IF (RCS_ON$(1:) = ON) and (RCS_ON$(3:) = ON) THEN
    --         RCS_ON$(1:),RCS_ON$(3:) = OFF;
    -- IF (RCS_ON$(2:) = ON) and (RCS_ON$(4:) = ON) THEN
    --         RCS_ON$(2:),RCS_ON$(4:) = OFF;
  end JET_SELECT_LOGIC_PACKAGE;
--*********************************--
  use BETA_FILTER_PACKAGE;
  use AERO_ANGLE_EXTRACT_PACKAGE;
  use TRANS_DELAY_COMP_PACKAGE;
  use STAB_AXES_CMD_PACKAGE;
  use JET_SELECT_LOGIC_PACKAGE;
--*********************************--

  procedure AERO_DAP        is

    ----------------------
    -- LOCAL PROCEDURES --
    ----------------------

      procedure AERO_DAP_INIT;

      procedure BODY_AXES_CMD;
```

```
procedure BODY_AXES_CMD    is
begin
----------------------
-- DAP ROLL CHANNEL --
----------------------
P_CMD := (BANK_RATE_CMD * CALPHA) * (BETA_RATE_CMD * SALPHA);
P_ERROR := P_CMD - P_FCS;
DP_CMD := K_P * P_ERROR;
-----------------------
-- DAP PITCH CHANNEL --
-----------------------
ALPHA_TRIM_CMD := ALPHA_CMD - TRIM_ERROR_L;
ALPHA_TRIM_ERROR := ALPHA_TRIM_CMD - ALPHA_DAP;
ALPHA_TRIM_ERROR_L := MIDVAL( -ALPHA_ERROR_LIM,ALPHA_TRIM_ERROR,
   ALPHA_ERROR_LIM);
Q_CMD := K_ALPHA * ALPHA_TRIM_ERROR_L;
Q_ERROR := Q_CMD - Q_FCS;
DQ_CMD := K_Q * Q_ERROR;
TRIM_ERROR_L := TRIM_ERROR_L * (K_ALPHA_TRIM * Q_ERROR * DT_AERODAP)
   ;
TRIM_ERROR_L := MIDVAL( -TRIM_ERROR_LIM,TRIM_ERROR_L,TRIM_ERROR_LIM)
   ;
----------------------
-- DAP YAW CHANNEL --
----------------------
R_CMD := (BETA_RATE_CMD * CALPHA) * (BANK_RATE_CMD * SALPHA);
R_ERROR := R_CMD - R_FCS;
DR_CMD := K_R * R_ERROR;
end BODY_AXES_CMD;
procedure AERO_DAP_INIT    is
begin
------------------
-- COPY I-LOADS --
------------------
DP1 := DP1_AERO;
DQ1 := DQ1_AERO;
DR1 := DR1_AERO;
DP2 := DP2_AERO;
DQ2 := DQ2_AERO;
DR2 := DR2_AERO;
DQ3 := DQ3_AERO;
DR3 := DR3_AERO;
DQ4 := DQ4_AERO;
DR4 := DR4_AERO;
DQ5 := DQ5_AERO;
DR5 := DR5_AERO;
DQ6 := DQ6_AERO;
DR6 := DR6_AERO;
end AERO_DAP_INIT;
begin
--****************************--
-- BODY OF PROCEDURE AERO_DAP --
--****************************--
------------------------
-- AERO_DAP EXECUTIVE --
------------------------
if FIRST_PASS then
   AERO_DAP_INIT;
```

```
    FIRST_PASS := FALSE;
  end if;
  AERO_ANGLE_EXTRACT;
  BETA_FILTER;
  TRANS_DELAY_COMP;
  STAB_AXES_CMD;
  BODY_AXES_CMD;
  JET_SELECT_LOGIC;
  ------------------------------------------------------
  -- COPY CYCLES FOR PLOTTING IN EDITOR - NOT DAP CODE --
  ------------------------------------------------------
  ------------------------
  -- GENERAL VARIABLES --
  ------------------------
  ALPHA_EDIT := ALPHA_DAP;
  BANK_RATE_CMD_EDIT := BANK_RATE_CMD;
  BETA_EDIT := BETA_DAP;
  BETA_FCS_EDIT := BETA_FCS;
  BETA_RATE_CMD_EDIT := BETA_RATE_CMD;
  PHI_EDIT := PHI_DAP;
  PHI_ERROR_EDIT := PHI_ERROR;
  ------------------------------------------
  -- TRANSPORT DELAY COMPENSATED BODY RATES --
  ------------------------------------------
  BODY_RATE_FCS_EDIT(1) := P_FCS;
  BODY_RATE_FCS_EDIT(2) := Q_FCS;
  BODY_RATE_FCS_EDIT(3) := R_FCS;
  ---------------
  -- ROLL AXIS --
  ---------------
  ATT_ERROR_EDIT(1) := PHI_ERROR;
  DP_CMD_EDIT := DP_CMD;
  P_ERROR_EDIT := P_ERROR;
  PC_EDIT := P_CMD;
  ----------------
  -- PITCH AXIS --
  ----------------
  ALPHA_TRIM_CMD_EDIT := ALPHA_TRIM_CMD;
  ALPHA_TRIM_ERROR_EDIT := ALPHA_TRIM_ERROR;
  ALPHA_TRIM_RATE_EDIT := ALPHA_TRIM_RATE;
  ATT_ERROR_EDIT(2) := ALPHA_TRIM_ERROR_L;
  DQ_CMD_EDIT := DQ_CMD;
  Q_ERROR_EDIT := Q_ERROR;
  QC_EDIT := Q_CMD;
  TRIM_ERROR_L_EDIT := TRIM_ERROR_L;
  --------------
  -- YAW AXIS --
  --------------
  ATT_ERROR_EDIT(3) :=  -BETA_FCS;
  DR_CMD_EDIT := DR_CMD;
```

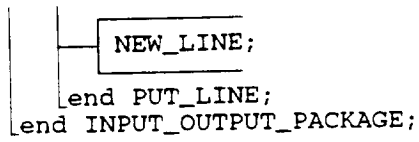
```
    |   |-- R_ERROR_EDIT := R_ERROR;
    |   |-- RC_EDIT := R_CMD;
    |   |--------------------
    |   |-- JSL VARIABLES --
    |   |--------------------
    |   |-- KP_RCS_EDIT := KP_RCS;
    |   |-- KQ_RCS_EDIT := KQ_RCS;
    |   |-- KR_RCS_EDIT := KR_RCS;
    |   |end AERO_DAP;
    |end AERO_DAP_PACKAGE;
```

```
with system;
use system;
with component_types;
use component_types;
with logical;
use logical;
with b1553_bc;
use b1553_bc;
with unchecked_conversion;

package body B1553_COMPONENT_DATA is

   data: arr_64;
   data_msg: arr_64;
   DATA_MSG2: ARR_64;
   stat_arr: arr1;
   msg_count: integer;
--  A_cmd:     UNSIGNED_WORD;
--  A_cmdlbk:  UNSIGNED_WORD;
--  A_stat:    UNSIGNED_WORD;
   msg_arr: arr_59_65;
   nmsg: integer;
   wdcount: arr_32;
   bc_interrupt_status: unsigned_word := 16#75#;
--  package int_io is new INTEGER_IO(INTEGER);
--  use int_io;
-------------------------------------------------------------------------

   procedure B1553_IMU_INTRP  is

   begin
   ----------------------------------------------------------------------------
   -- Message 1 --
   -- Set up IMU 40 msec interrupt - Data Ready Signal --
   bc_interrupt_status := unsigned_word(16#75#);
   while  (short_and(bc_interrupt_status,16#74#) /= 16#0000#) loop
      data_msg(1) := 16#0001#;
      -- Even and Odd frame data  --
      data_msg(2) := 16#1000#;
      -- BIT 12 DATA READY SIGNAL - 40 MSEC  --
      data_msg(3) := 16#0000#;

      bc_store_msg(0,2,3,0,3,data_msg);

      -- Data word - RT 2 Subadd 3  --
      -- rcv 3 data words           --

      BC_GO;

      BC_INTERRUPT(bc_interrupt_status);

   end loop;
   -- Wait for BC interrupt then --
   -- change buffer  --
   --   put(" bc_interrupt_status = ");
   --   put(integer(bc_interrupt_status),4,16);
   --   new_line;
   end B1553_IMU_INTRP;
-- end bc_interrupt_status loop --
-- Timeout/1553 format error; buffer overflow;--
-- loop test fail; status set --
-- End Message 1 --
```

```
-------------------------------------------------------------------------
-------------------------------------------------------------------------

   procedure B1553_IMU_INIT  is

   begin
   ---------------------------------------------------------------------
   -- Message 2 --
   -- Set up IMU Quaternion Initialization  --
   -- bc_interrupt_status := unsigned_word(16#75#);
   -- while  (short_and(bc_interrupt_status,16#74#) /= 16#0000#) loop
   --    data_msg(1) := 16#0001#;
   --    -- Even and Odd frame data  --
   --    data_msg(2) := 16#1002#;
   --    -- BIT 12 DATA READY SIGNAL, BIT 1 RESET --
   --    --   QUATERNION TO (1,0,,0,0)  --
   --    data_msg(3) := 16#0000#;
   --
   --    bc_store_msg(0,2,3,0,3,data_msg);
   --
   --    -- Data word - RT 2 Subadd 3  --
   --    -- rcv 3 data words           --
   --
   --    BC_GO;
   --
   --    BC_INTERRUPT(bc_interrupt_status);
   --
   -- end loop;
   -- Wait for BC interrupt then --
   -- change buffer  --
   --   put(" bc_interrupt_status = ");
   --   put(integer(bc_interrupt_status),4,16);
   --   new_line;
   end B1553_IMU_INIT;
-- end bc_interrupt_status loop --
-- Timeout/1553 format error; buffer overflow;--
-- loop test fail; status set --
-- End Message 2 --
-------------------------------------------------------------------------
-------------------------------------------------------------------------

   procedure READ_IMU_DATA(IMU_DATA: out ARR_32)    is

   begin
   -- bc_interrupt_status := unsigned_word(16#75#);
   -- while (short_and(bc_interrupt_status,16#74#) /= 16#0000#) loop
   --
   --    bc_store_msg(0,2,2,1,32,data_msg);
   --
   --    -- Data word - Rt 2 Subaddr 2 --
   --    -- xmit 32 data words         --
   --    -- EVEN Frame Data - Subaddr 2 --
   --
   --    bc_go;
   --
   --    bc_interrupt(bc_interrupt_status);
   --
   -- end loop;
   --   put(" bc_interrupt_status = ");
   --   put(integer(bc_interrupt_status),4,16);
   --   new_line;
```

```
    -- end bc_interrupt_status loop --
    -- Timeout/1553 format error; buffer overflow;--
    -- loop test fail; status set --
    ------------------------------------------------------------------------
    --  BC_status(A_cmd,A_cmdlbk,A_stat,1);
    --   put (" A_cmd = ");      put(integer(A_cmd),4,16);
    --   put (" A_cmdlbk = ");   put(integer(A_cmdlbk),4,16);
    --   put (" A_stat = ");     put(integer(A_stat),4,16);
    --   new_line;
    BC_get_msg(msg_arr);
    --      msg_count := integer(msg_arr(1,1));
    --    put (" Message count = ");
    --    put(msg_count,4,16);
    --    new_line;
    --    put (" Message = ");
    --    new_line;
    for i in 1..32 loop
      imu_data(i) := msg_arr(1,i + 1);
    end loop;
  end READ_IMU_DATA;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------
  procedure THRUSTER_INIT  is
  begin
  -- Clear thrusters in Message 2 --
  data_msg2(1) := 16#0000#;
  data_msg2(2) := 16#0000#;
  data_msg2(3) := 16#0000#;
  ------------------------------------------------------------------------
  ------------------------------------------------------------------------
  ------------------------------------------------------------------------
  -- THRUSTERS INITIALIZED TO ALL OFF CONDITION  ---------
  bc_interrupt_status := unsigned_word(16#75#);
  while (short_and(bc_interrupt_status,16#74#) /= 16#0000#) loop
    bc_store_msg(0,3,2,0,3,data_msg2);
    -- Data word - Rt 3 Subaddr 2 --
    -- rcv  3 data words          --
    bc_go;
    bc_interrupt(bc_interrupt_status);
  end loop;
  end THRUSTER_INIT;
-- end bc_interrupt_status loop --
-- Timeout/1553 format error; buffer overflow;--
-- loop test fail; status set --
-- End Message 2 --
end B1553_COMPONENT_DATA;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
```

```
package body INPUT_OUTPUT_PACKAGE is

   use SCALAR_SINGLE_IO;
   use SCALAR_DOUBLE_IO;

   procedure PUT_LINE (X: SINGLE_PRECISION_VECTOR) is

   begin
      for I in X'FIRST..X'LAST loop
         PUT(X(I));
      end loop;
      NEW_LINE;
   end PUT_LINE;

   procedure PUT_LINE (X: DOUBLE_PRECISION_VECTOR) is

   begin
      for I in X'FIRST..X'LAST loop
         PUT(X(I));
      end loop;
      NEW_LINE;
   end PUT_LINE;

   procedure PUT_LINE (MAT: SINGLE_PRECISION_MATRIX) is

   begin
      for I in MAT'FIRST(1)..MAT'LAST(1) loop
         for J in MAT'FIRST(2)..MAT'LAST(2) loop
            PUT(MAT(I,J));
         end loop;
         NEW_LINE;
      end loop;
      NEW_LINE;
   end PUT_LINE;

   procedure PUT_LINE (MAT: DOUBLE_PRECISION_MATRIX) is

   begin
      for I in MAT'FIRST(1)..MAT'LAST(1) loop
         for J in MAT'FIRST(2)..MAT'LAST(2) loop
            PUT(MAT(I,J));
         end loop;
         NEW_LINE;
      end loop;
```


NEW_LINE;
end PUT_LINE;
end INPUT_OUTPUT_PACKAGE;

```
with LEVEL_A_CONSTANTS;
use LEVEL_A_CONSTANTS;
with DATA_TYPES;
use DATA_TYPES;
with FSW_POOL;
use FSW_POOL;
with IL_POOL;
use IL_POOL;
with TEXT_IO;
use TEXT_IO;
with INPUT_OUTPUT_PACKAGE;
use INPUT_OUTPUT_PACKAGE;
with MATH_PACKAGE;
use MATH_PACKAGE;
with QUATERNION_OPERATIONS;
use QUATERNION_OPERATIONS;
with SINGLE_PRECISION_MATRIX_OPERATIONS;
use SINGLE_PRECISION_MATRIX_OPERATIONS;
with DOUBLE_PRECISION_MATRIX_OPERATIONS;
use DOUBLE_PRECISION_MATRIX_OPERATIONS;

package body PRED_GUID_PACKAGE is

   APOGEE_EPSILON1 : SCALAR_SINGLE := 25.0;
-------------------------------------------------------------
-- FUNCTION: NUMERIC PREDICTOR/CORRECTOR AEROBRAKING GUIDANCE --
-------------------------------------------------------------
-----------------------------------------------------
-- ILOADS - MOVE TO ILPOOL IF RETAIN THIS ALGORITHM? --
-----------------------------------------------------
   APOGEE_EPSILON2 : SCALAR_SINGLE := 1.0;
   BANK_MAX : SCALAR_SINGLE := 165.0;
   BANK_MIN : SCALAR_SINGLE := 15.0;
   CORRIDOR_MIN :  constant SCALAR_SINGLE := 0.05;
   CORRIDOR_V_MAX :  constant SCALAR_SINGLE := 34_000.0;
   CORRIDOR_V_MIN :  constant SCALAR_SINGLE := 26500.0;
   DELTA_PHI_MIN : SCALAR_SINGLE := 1.0;
   DELTA_T_PRED :  constant SCALAR_SINGLE := 2.0;
   G_RUN_GUIDANCE : SCALAR_SINGLE := 0.075;
   GUID_PASS_LIM :  constant INTEGER := 10;
   LIFT_INC_CAPTURE : SCALAR_SINGLE := 0.15;
   LIFT_PERCENT_CAPTURE : SCALAR_SINGLE := 0.5;
   MAX_NUMBER_RUNS :  constant INTEGER := 5;
   PHI_LIFT_DOWN :  constant SCALAR_SINGLE := 45.0;
   VI_LIFT_DOWN :  constant SCALAR_SINGLE := 27500.0;
   VI_MODEL_LIFT_DOWN :  constant SCALAR_SINGLE := 27900.0;
   COS_PHI_MAX : SCALAR_SINGLE := 0.0;
---------------------
-- LOCAL VARIABLES --
---------------------
   COS_PHI_MIN : SCALAR_SINGLE := 0.0;
   GUID_PASS : INTEGER := 0;
   INITIALIZE_GUIDANCE : BOOLEAN_32 := TRUE;
   MODEL_LIFT_DOWN : BOOLEAN_32 := TRUE;
   PHI_CMD_NS : SCALAR_SINGLE := 0.0;
   SIGN_OF_BANK : SCALAR_SINGLE := 0.0;
   FIRST_TIME_CALLED : BOOLEAN_32 := TRUE;
   EARTH_POLE : DOUBLE_PRECISION_VECTOR3 := (others=>0.0);
   EARTH_OMEGA : DOUBLE_PRECISION_VECTOR3 := (others=>0.0);
   ZERO :  constant SCALAR_SINGLE := 0.0;
-------------------------------------------------------------
-- NUMERICAL CONSTANTS USED IN PACKAGE                     --
-- This is necessary because of the overloading of operator --
```

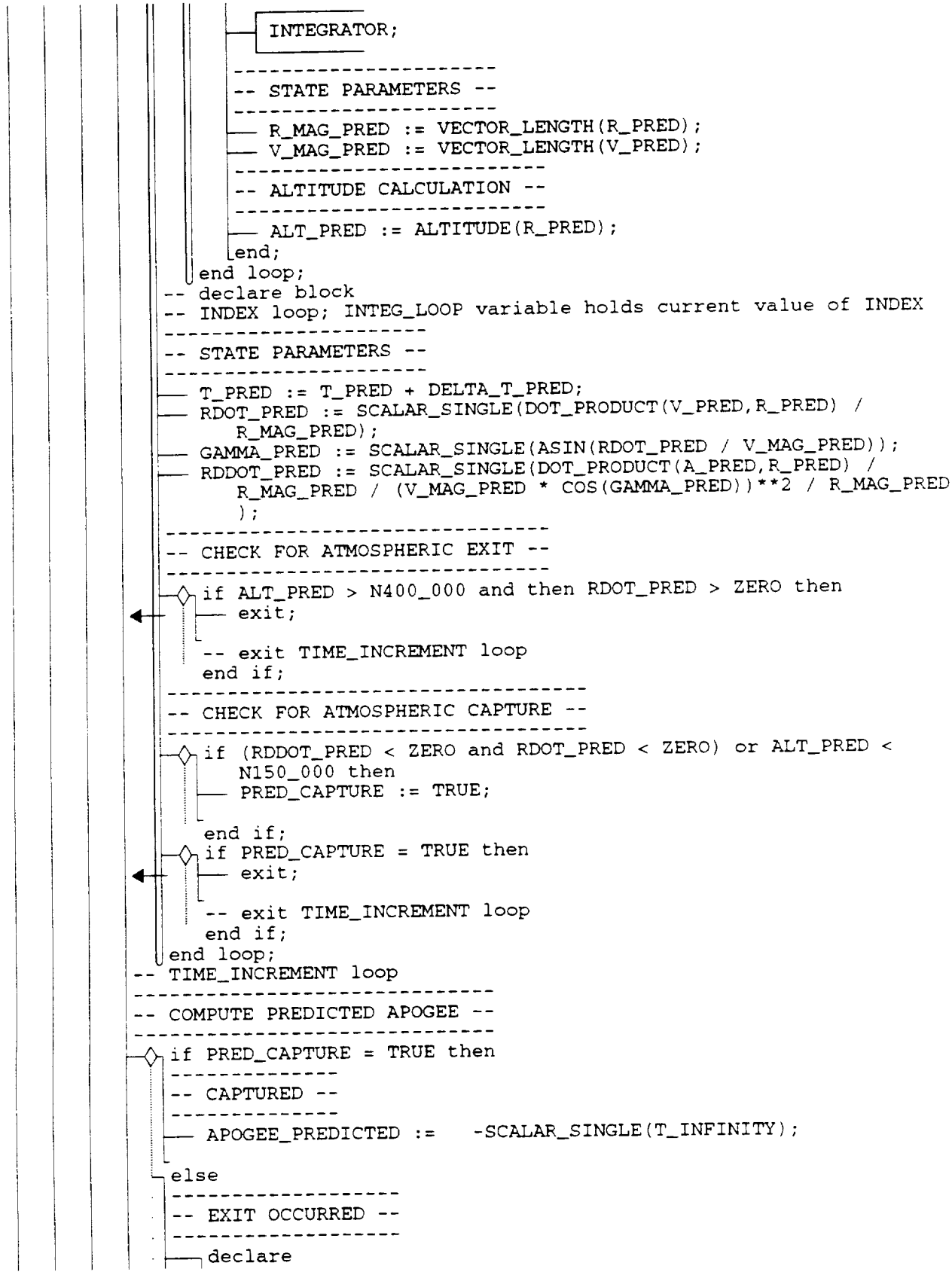
```
-- symbols to allow mixed mode arithmetic between single-    --
-- precision and double-precision variables.                 --
---------------------------------------------------------------
   ONE_TENTH :  constant SCALAR_SINGLE := 0.1;
   ONE_HALF :  constant SCALAR_SINGLE := 0.5;
   ONE:  constant SCALAR_SINGLE := 1.0;
   TWO :  constant SCALAR_SINGLE := 2.0;
   THREE :  constant SCALAR_SINGLE := 3.0;
   FIVE :  constant SCALAR_SINGLE := 5.0;
   N25_000 :  constant SCALAR_SINGLE := 25000.0;
   N26_000 :  constant SCALAR_SINGLE := 26000.0;
   N27_000 :  constant SCALAR_SINGLE := 27000.0;
   N29_000 :  constant SCALAR_SINGLE := 29000.0;
   N30_000 :  constant SCALAR_SINGLE := 30000.0;
   N33_850 :  constant SCALAR_SINGLE := 33850.0;
   N150_000 :  constant SCALAR_SINGLE := 150_000.0;
   N400_000 :  constant SCALAR_SINGLE := 400_000.0;
-------------------------------------------------
-- USE OUTPUT ROUTINES FROM INPUT_OUTPUT_PACKAGE --
-------------------------------------------------
   use INPUT_OUTPUT_PACKAGE.INT_IO;
   use INPUT_OUTPUT_PACKAGE.SCALAR_SINGLE_IO;
-------------------------------------------------------------------
-- USE A MATH PACKAGE TAILORED TO PROVIDE THE PRECISION WE NEED --
-- FOR THIS APPLICATION                                         --
-------------------------------------------------------------------
   use SINGLE_PRECISION_MATRIX_OPERATIONS.REAL_MATH_LIB;
   use DOUBLE_PRECISION_MATRIX_OPERATIONS.REAL_MATH_LIB;
--------------------
-- LOCAL FUNCTION --
--------------------
   function ALTITUDE (R: DOUBLE_PRECISION_VECTOR3) return SCALAR_DOUBLE ;
-------------------------------------------------------------------
-- THE FOLLOWING PACKAGES CONTAIN PROCEDURES THAT ARE CALLED BY   --
-- procedure PRED_GUID. THEY ARE POSITIONED EXTERNAL TO procedure --
-- PRED_GUID SO THAT THEIR VARIABLES WILL EXIST BEYOND THE TIME   --
-- WHEN THE PROCEDURE IS EXECUTING.                               --
-------------------------------------------------------------------
   package PC_SEQUENCER_PACKAGE   is

      procedure PC_SEQUENCER;

   end PC_SEQUENCER_PACKAGE;

   package LATERAL_CONTROL_PACKAGE   is

      procedure LATERAL_CONTROL;

   end LATERAL_CONTROL_PACKAGE;
   use PC_SEQUENCER_PACKAGE;
   use LATERAL_CONTROL_PACKAGE;
--------------------------------------
-- BODY OF FUNCTION SPECIFIED ABOVE --
--------------------------------------
--*****************************************************************--

   function ALTITUDE (R: DOUBLE_PRECISION_VECTOR3) return SCALAR_DOUBLE  is
```

```
   --*************************************************************************--
      RM : SCALAR_DOUBLE;
   begin
   ------------------------------------------------
   -- COMPUTES THE ALTITUDE ABOVE FISCHER ELLIPSOID --
   ------------------------------------------------
   -- RM := VECTOR_LENGTH(R);
   -- return (RM / EARTH_R - (ONE - EARTH_FLAT) / SQRT(ONE / ((ONE -
         EARTH_FLAT)**2 - ONE) / (ONE / (DOT_PRODUCT((R / RM),EARTH_POLE)**2)
         )));
   end ALTITUDE;
-------------------------------------  BODIES OF PACKAGES SPECIFIED ABOVE
---- ------------------------------------ *********************************
--*************************************************************************
--********--

   package body PC_SEQUENCER_PACKAGE is

   ------------------------------------------------
   -- LOCAL VARIABLES - POSITIONED HERE FOR DUMP --
   ------------------------------------------------
      APOGEE_BRACKET : array(1..2) of SCALAR_SINGLE;
      APOGEE_EPSILON : SCALAR_SINGLE;
      APOGEE_EXTRAPOLATE : array(1..2) of SCALAR_SINGLE;
      APOGEE_PREDICTED : SCALAR_SINGLE;
      BRACKETED : BOOLEAN_32;
      COS_CAPT : SCALAR_SINGLE;
      COS_BRACKET : array(1..2) of SCALAR_SINGLE;
      COS_EXTRAPOLATE : array(1..2) of SCALAR_SINGLE;
      COS_PHI_TRY : array(1..10) of SCALAR_SINGLE;
      DELTA_APOGEE : SCALAR_SINGLE;
      DELTA_PHI : SCALAR_SINGLE;
      I : INTEGER;
      INTEG_LOOP : INTEGER range 1..4;
      NUMBER_CAPT : INTEGER;
      NUMBER_GOOD : INTEGER;
      NUMBER_HIGH : INTEGER;
      NUMBER_LOW : INTEGER;
      PHI_TRY : SCALAR_SINGLE;
      PHI_TRY_LAST : SCALAR_SINGLE;
      PRED_CAPTURE : BOOLEAN_32;
   ------------------------------------------------------------------------
   -- LOCAL PROCEDURES CALLED BY procedure PC_SEQUENCER.                 --
   -- APPEAR HERE IN PACKAGE FORMAT SO THAT VARIABLES WILL BE AVAILABLE  --
   -- FOR DUMPS AND SO THAT VARIABLE VALUES WILL EXIST BETWEEN INVOCATIONS --
   -- OF THESE PROCEDURES BY procedure PC_SEQUENCER.                     --
   ------------------------------------------------------------------------

      package PREDICTOR_PACKAGE   is

         procedure PREDICTOR;

      end PREDICTOR_PACKAGE;

      package CORRECTOR_PACKAGE   is

         procedure CORRECTOR;

      end CORRECTOR_PACKAGE;
      use PREDICTOR_PACKAGE;
      use CORRECTOR_PACKAGE;
```

```
--*****************************************************************************
--*********************************************--

  package body PREDICTOR_PACKAGE is

  ------------------------------------------------
  -- LOCAL TO PREDICTOR - POSITIONED HERE FOR DUMP --
  ------------------------------------------------
     A_PRED : DOUBLE_PRECISION_VECTOR3;
     ALT_PRED : SCALAR_DOUBLE;
     GAMMA_PRED : SCALAR_SINGLE;
     LOD_PRED : SCALAR_SINGLE;
     PHI_PRED : SCALAR_SINGLE;
     R_PRED : DOUBLE_PRECISION_VECTOR3;
     R_MAG_PRED : SCALAR_DOUBLE;
     RDDOT_PRED : SCALAR_SINGLE;
     RDOT_PRED : SCALAR_SINGLE;
     T_PRED : SCALAR_DOUBLE;
     V_MAG_PRED : SCALAR_DOUBLE;
     V_PRED : DOUBLE_PRECISION_VECTOR3;
  ------------------------------------------------------------------------
  -- INTEGRATOR PROCEDURE CALLED BY procedure PREDICTOR.                 --
  -- APPEARS HERE AS A PACKAGE SO THAT ITS VARIABLES WILL RETAIN         --
  -- THEIR VALUES BETWEEN INVOCATIONS OF THE PROCEDURE BY PREDICTOR.     --
  ------------------------------------------------------------------------

     package INTEGRATOR_PACKAGE   is

        procedure INTEGRATOR;

     end INTEGRATOR_PACKAGE;
  --*****************************************************************************
  --*********************************************--

     package body INTEGRATOR_PACKAGE is

     ------------------------------------------------------------------------
     ----  VARIABLES ARE DECLARED AND POSITIONED HERE SO THAT THEIR VALUE
     --S  --  WILL EXIST FROM INVOCATION TO INVOCATION OF procedure INTEG
     --RATOR -- ---------------------------------------------------------
     -------------
        ACCUM_ACCEL : DOUBLE_PRECISION_VECTOR3;
        ACCUM_VEL : DOUBLE_PRECISION_VECTOR3;
        ORIG_POS : DOUBLE_PRECISION_VECTOR3;
        ORIG_VEL : DOUBLE_PRECISION_VECTOR3;
     --********************--

        procedure INTEGRATOR is

        --********************--
        begin
           case INTEG_LOOP is
              when 1 =>
                 ORIG_POS := R_PRED;
                 ORIG_VEL := V_PRED;
                 ACCUM_VEL := V_PRED;
                 ACCUM_ACCEL := A_PRED;
                 R_PRED := ORIG_POS * ONE_HALF * DELTA_T_PRED * V_PRED;
                 V_PRED := ORIG_VEL * ONE_HALF * DELTA_T_PRED * A_PRED;

              when 2 =>
                 ACCUM_VEL := ACCUM_VEL * TWO * V_PRED;
```

```
                    ACCUM_ACCEL := ACCUM_ACCEL * TWO * A_PRED;
                    R_PRED := ORIG_POS * ONE_HALF * DELTA_T_PRED * V_PRED;
                    V_PRED := ORIG_VEL * ONE_HALF * DELTA_T_PRED * A_PRED;
                when 3 =>
                    ACCUM_VEL := ACCUM_VEL * TWO * V_PRED;
                    ACCUM_ACCEL := ACCUM_ACCEL * TWO * A_PRED;
                    R_PRED := ORIG_POS * DELTA_T_PRED * V_PRED;
                    V_PRED := ORIG_VEL * DELTA_T_PRED * A_PRED;
                when 4 =>
                    R_PRED := ORIG_POS / (ACCUM_VEL + V_PRED) * DELTA_T_PRED
                        / 6.0;
                    V_PRED := ORIG_VEL / (ACCUM_ACCEL + A_PRED) *
                        DELTA_T_PRED / 6.0;
                when others  =>
                    -- INTEG_LOOP can only have values in the range 1..4
                    null;
                end case;
            end INTEGRATOR;
        end INTEGRATOR_PACKAGE;
        use INTEGRATOR_PACKAGE;
    --********************--

    procedure PREDICTOR is

        --********************--
        begin
        ---------------------------------------
        -- INITIALIZE PREDICTOR STATE VECTOR --
        ---------------------------------------
        R_PRED := R_NAV;
        R_MAG_PRED := VECTOR_LENGTH(R_PRED);
        ALT_PRED := ALTITUDE(R_PRED);
        V_PRED := V_NAV;
        V_MAG_PRED := VECTOR_LENGTH(V_PRED);
        PHI_PRED := PHI_TRY * SIGN_OF_BANK;
        T_PRED := T_GMT;
        LOD_PRED := CL_NAV / CD_NAV;
        PRED_CAPTURE := FALSE;
        --------------------
        -- PREDICTOR LOOP --
        --------------------
        for TIME_INCREMENT in 1..750 loop
            ------------------------------------------
            -- 4TH ORDER RUNGA_KUTTA INTEGRATION LOOP --
            ------------------------------------------
            for INDEX in 1..4 loop
                INTEG_LOOP := INDEX;
                declare
                    AERO_ACCEL : DOUBLE_PRECISION_VECTOR3;
                    ALT_NORM_PRED : SCALAR_SINGLE;
                    CPHI : SCALAR_SINGLE;
                    DRAG_ACCEL : SCALAR_SINGLE;
                    GRAV_ACCEL : DOUBLE_PRECISION_VECTOR3;
                    HS_NORM_PRED : SCALAR_SINGLE;
                    I_LAT : DOUBLE_PRECISION_VECTOR3;
                    I_LIFT : DOUBLE_PRECISION_VECTOR3;
                    I_VEL : DOUBLE_PRECISION_VECTOR3;
                    LIFT_ACCEL : SCALAR_SINGLE;
                    RHO_EST : SCALAR_SINGLE;
```

```
   RHO_NOM : SCALAR_SINGLE;
   SPHI : SCALAR_SINGLE;
   U_PRED : DOUBLE_PRECISION_VECTOR3;
   V_REL_MAG_PRED : SCALAR_DOUBLE;
   V_REL_PRED : DOUBLE_PRECISION_VECTOR3;
   Z_PRED : SCALAR_DOUBLE;
begin
-----------------------
-- RELATIVE VELOCITY --
-----------------------
V_REL_PRED := V_PRED - CROSS_PRODUCT(EARTH_OMEGA,R_PRED)
   ;
V_REL_MAG_PRED := VECTOR_LENGTH(V_REL_PRED);
---------------------------------------
-- 1962 STANDARD ATMOSPHERE CURVE FIT --
---------------------------------------
ALT_NORM_PRED := SCALAR_SINGLE(ALT_PRED / H_REF);
HS_NORM_PRED :=  (((C_HS(5) * ALT_NORM_PRED + C_HS(4)) *
   ALT_NORM_PRED + C_HS(3)) * ALT_NORM_PRED + C_HS(2)) *
   ALT_NORM_PRED + C_HS(1);
RHO_NOM := RHO_REF / EXP((ONE - ALT_NORM_PRED) /
   HS_NORM_PRED);
-----------------------
-- ESTIMATED DENSITY --
-----------------------
RHO_EST := K_RHO_NAV * RHO_NOM;
--------------------
-- LIFTDOWN MODEL --
--------------------
if MODEL_LIFT_DOWN = TRUE and V_MAG_PRED < VI_LIFT_DOWN
   then
   PHI_PRED := PHI_LIFT_DOWN * SIGN_OF_BANK;
end if;
CPHI := COS(PHI_PRED * DEG_TO_RAD);
SPHI := SIN(PHI_PRED * DEG_TO_RAD);
------------------------------
-- AERODYNAMIC ACCELERATIONS --
------------------------------
DRAG_ACCEL := SCALAR_SINGLE((ONE_HALF * RHO_EST *
   V_REL_MAG_PRED**2 * CD_NAV * S_REF) / MASS_NAV);
LIFT_ACCEL := LOD_PRED * DRAG_ACCEL;
I_VEL := V_REL_PRED / V_REL_MAG_PRED;
I_LAT := UNIT(CROSS_PRODUCT(I_VEL,R_PRED));
I_LIFT := UNIT(CROSS_PRODUCT(I_LAT,I_VEL)) * CPHI *
   I_LAT * SPHI;
AERO_ACCEL := LIFT_ACCEL * I_LIFT * DRAG_ACCEL * I_VEL;
-------------------------------------
-- GRAVITY ACCELERATION WITH J2 TERM --
-------------------------------------
U_PRED := R_PRED / R_MAG_PRED;
Z_PRED := DOT_PRODUCT(U_PRED,EARTH_POLE);
U_PRED := U_PRED * (THREE * EARTH_J2 / TWO) / (EARTH_R /
   R_MAG_PRED)**2 * ((ONE * FIVE * Z_PRED**2) * U_PRED *
   TWO * Z_PRED * EARTH_POLE);
GRAV_ACCEL :=  -(EARTH_MU / R_MAG_PRED**2) * U_PRED;
------------------------
-- TOTAL ACCELERATION --
------------------------
A_PRED := AERO_ACCEL + GRAV_ACCEL;
--------------------------------
-- CALL RUNGA_KUTTA INTEGRATOR --
--------------------------------
```

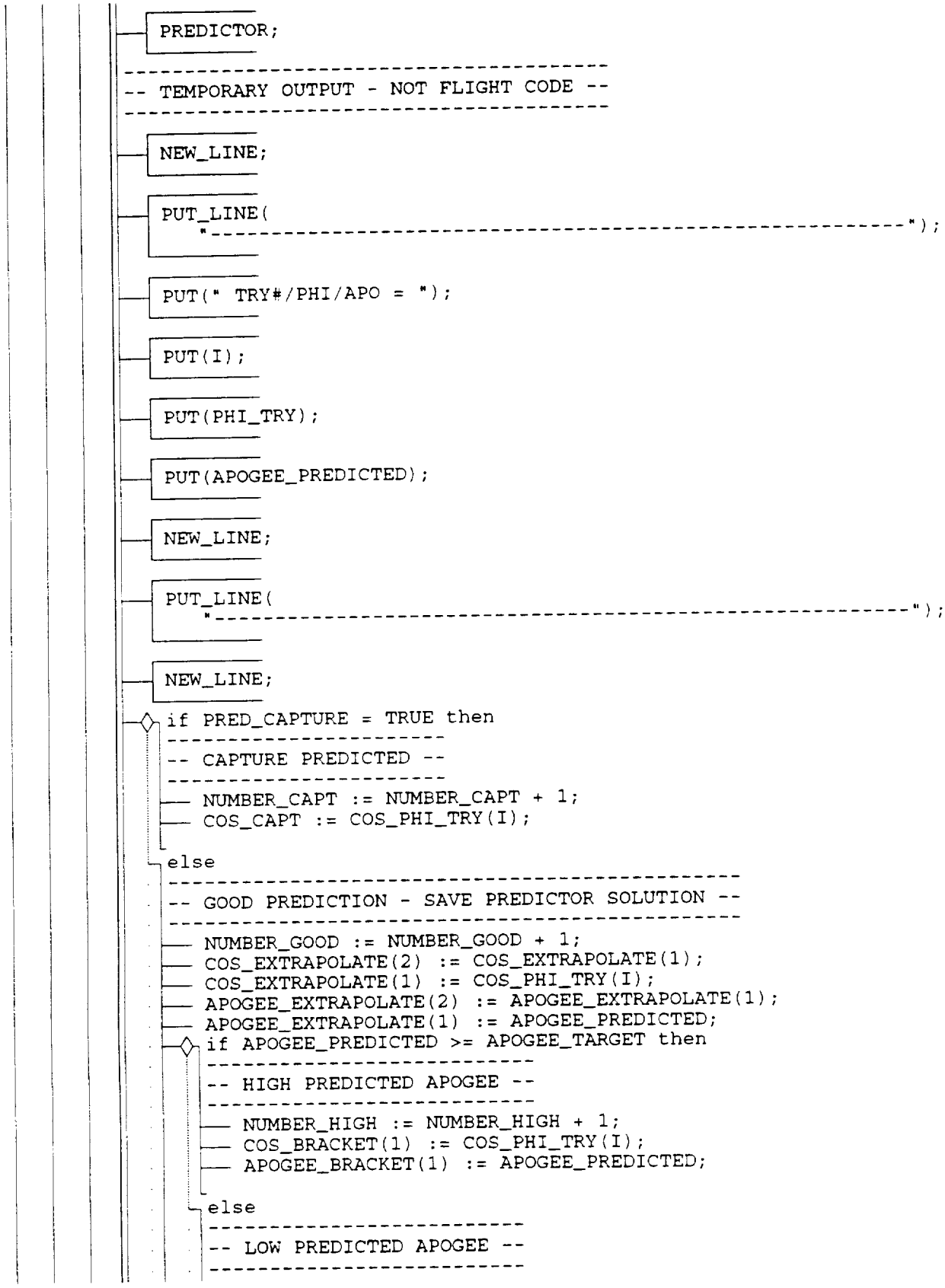


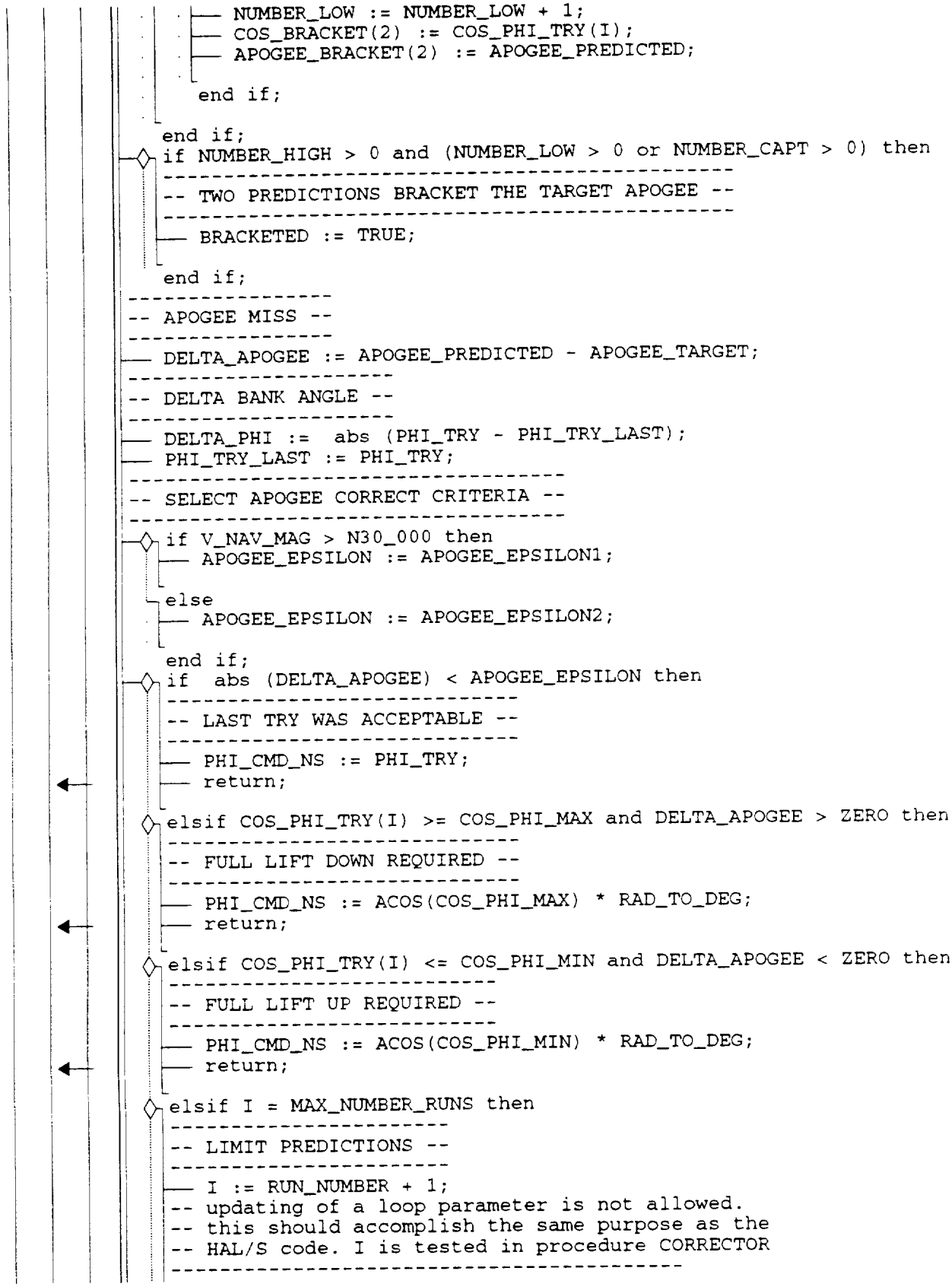
```
            INTEGRATOR;
            ----------------------
            -- STATE PARAMETERS --
            ----------------------
            R_MAG_PRED := VECTOR_LENGTH(R_PRED);
            V_MAG_PRED := VECTOR_LENGTH(V_PRED);
            --------------------------
            -- ALTITUDE CALCULATION --
            --------------------------
            ALT_PRED := ALTITUDE(R_PRED);
          end;
        end loop;
      -- declare block
      -- INDEX loop; INTEG_LOOP variable holds current value of INDEX
      ----------------------
      -- STATE PARAMETERS --
      ----------------------
      T_PRED := T_PRED + DELTA_T_PRED;
      RDOT_PRED := SCALAR_SINGLE(DOT_PRODUCT(V_PRED,R_PRED) /
          R_MAG_PRED);
      GAMMA_PRED := SCALAR_SINGLE(ASIN(RDOT_PRED / V_MAG_PRED));
      RDDOT_PRED := SCALAR_SINGLE(DOT_PRODUCT(A_PRED,R_PRED) /
          R_MAG_PRED / (V_MAG_PRED * COS(GAMMA_PRED))**2 / R_MAG_PRED
          );
      --------------------------------
      -- CHECK FOR ATMOSPHERIC EXIT --
      --------------------------------
      if ALT_PRED > N400_000 and then RDOT_PRED > ZERO then
        exit;

        -- exit TIME_INCREMENT loop
      end if;
      -----------------------------------
      -- CHECK FOR ATMOSPHERIC CAPTURE --
      -----------------------------------
      if (RDDOT_PRED < ZERO and RDOT_PRED < ZERO) or ALT_PRED <
          N150_000 then
        PRED_CAPTURE := TRUE;

      end if;
      if PRED_CAPTURE = TRUE then
        exit;

        -- exit TIME_INCREMENT loop
      end if;
    end loop;
  -- TIME_INCREMENT loop
  ----------------------------
  -- COMPUTE PREDICTED APOGEE --
  ----------------------------
  if PRED_CAPTURE = TRUE then
    --------------
    -- CAPTURED --
    --------------
    APOGEE_PREDICTED :=    -SCALAR_SINGLE(T_INFINITY);

  else
    --------------------
    -- EXIT OCCURRED --
    --------------------
    declare
```

```
            ECCEN_PRED : SCALAR_SINGLE;
            PARAMETER_PRED : SCALAR_SINGLE;
         begin
            PARAMETER_PRED := SCALAR_SINGLE((R_MAG_PRED * V_MAG_PRED *
               COS(GAMMA_PRED))**2 / EARTH_MU);
            ECCEN_PRED := SCALAR_SINGLE(SQRT(ONE / PARAMETER_PRED / (
               TWO / R_MAG_PRED / V_MAG_PRED**2 / EARTH_MU)));
            APOGEE_PREDICTED := SCALAR_SINGLE((PARAMETER_PRED - (ONE -
               ECCEN_PRED) - EARTH_R) * FT_TO_NM);
         end;

      -- declare block
      end if;
   end PREDICTOR;
end PREDICTOR_PACKAGE;
--*****************************************************************************
--*******************************************--

package body CORRECTOR_PACKAGE is

   -------------------------------------------------
   -- LOCAL TO CORRECTOR - POSITIONED HERE FOR DUMP --
   -------------------------------------------------
      DELT : SCALAR_SINGLE;
      RISE : SCALAR_SINGLE;
      RUN : SCALAR_SINGLE;
      SENSITIVITY : SCALAR_SINGLE;
      TRY_METHOD : INTEGER range 1..6;
   --********************--

      procedure CORRECTOR is

      --********************--
      begin
      ---------------------------------------------
      -- COMPUTE PREFLIGHT PREDICTED SENSITIVITY --
      ---------------------------------------------
      if V_NAV_MAG > N33_850 then
         SENSITIVITY := 24000.0;

      elsif V_NAV_MAG > N30_000 then
         SENSITIVITY := SCALAR_SINGLE(SCALAR_DOUBLE(6.3926) * V_NAV_MAG
            - SCALAR_DOUBLE(188_700.0));

      elsif V_NAV_MAG > N29_000 then
         SENSITIVITY := SCALAR_SINGLE(SCALAR_DOUBLE(1.49013) *
            V_NAV_MAG - SCALAR_DOUBLE(41625.0));

      elsif V_NAV_MAG > N27_000 then
         SENSITIVITY := SCALAR_SINGLE(SCALAR_DOUBLE(0.57892) *
            V_NAV_MAG - SCALAR_DOUBLE(15200.0));

      elsif V_NAV_MAG > N26_000 then
         SENSITIVITY := SCALAR_SINGLE(SCALAR_DOUBLE(0.42596) *
            V_NAV_MAG - SCALAR_DOUBLE(11070.0));

      elsif V_NAV_MAG > N25_000 then
         SENSITIVITY := 5.0;

      end if;
      ------------------------------------
      -- DETERMINE WAY TO MAKE NEXT GUESS --
      ------------------------------------
```

```
-- I is declared in PC_SEQUENCER_PACKAGE and is set equal
-- to RUN_NUMBER in RUN_NUMBER loop
if I = 1 then
   TRY_METHOD := 1;

else
   if BRACKETED = TRUE then
      if NUMBER_LOW /= 0 then
         TRY_METHOD := 2;

      else
         TRY_METHOD := 3;

      end if;

   else
      case MIDVAL(0,NUMBER_GOOD,2) is
         when 1 =>
            TRY_METHOD := 5;

         when 2 =>
            TRY_METHOD :=6;

         when others  =>
            TRY_METHOD := 4;

      end case;

   end if;
end if;
case TRY_METHOD is
   when 1 =>
      ------------------------------------------------
      -- RUN LAST GUESS FROM PREVIOUS GUIDANCE CYCLE --
      ------------------------------------------------
      COS_PHI_TRY(I) := COS(PHI_CMD * DEG_TO_RAD);

   when 2 =>
      ------------------------------------------------------------
      ---  INTERPOLATE A HIGH GUESS AND A LOW GUESS TO TARGET APOGEE
      -- -- ---------------------------------------------------------
      -------
      RUN := COS_BRACKET(2) - COS_BRACKET(1);
      RISE := APOGEE_BRACKET(2) - APOGEE_BRACKET(1);
      if  abs (RISE) < ONE_TENTH then
         RISE := ONE_TENTH * SIGN(RISE);

      end if;
      DELT := APOGEE_TARGET - APOGEE_BRACKET(1);
      COS_PHI_TRY(I) := COS_BRACKET(1) / (DELT * RUN) / RISE;

   when 3 =>
      ------------------------------------------------
      -- INTERPOLATE A HIGH GUESS AND A CAPTURED GUESS --
      -- A % FROM HIGH GUESS                           --
      ------------------------------------------------
      COS_PHI_TRY(I) := COS_BRACKET(1) * (COS_CAPT - COS_BRACKET(
         1)) * LIFT_PERCENT_CAPTURE;

   when 4 =>
      ------------------------------------
      -- MARCH OUT OF THE CAPTURE REGION --
```

```
                    ------------------------------------
                    COS_PHI_TRY(I) := COS_CAPT - LIFT_INC_CAPTURE;

                when 5 =>
                    ------------------------------------------------------------
                    -- EXTRAPOLATE ONE GOOD GUESS USING A STORED SENSITIVITY --
                    ------------------------------------------------------------
                    COS_PHI_TRY(I) := COS_PHI_TRY(I - 1) / DELTA_APOGEE /
                        SENSITIVITY;

                when 6 =>
                    ------------------------------------------------------
                    -- EXTRAPOLATE TWO HIGH GUESSES OR TWO LOW GUESSES --
                    -- TO TARGET APOGEE                                --
                    ------------------------------------------------------
                    RUN := COS_EXTRAPOLATE(2) - COS_EXTRAPOLATE(1);
                    RISE := APOGEE_EXTRAPOLATE(2) - APOGEE_EXTRAPOLATE(1);
                    if  abs (RISE) < ONE_TENTH then
                        RISE := ONE_TENTH * SIGN(RISE);

                    end if;
                    DELT := APOGEE_TARGET - APOGEE_EXTRAPOLATE(1);
                    COS_PHI_TRY(I) := COS_EXTRAPOLATE(1) / (DELT * RUN) / RISE;

                when others  =>
                    -- TRY_METHOD can only have values from 1..6
                    null;

                end case;
            -------------------------
            -- NEW GUESS FOR PHI_TRY --
            -------------------------
            COS_PHI_TRY(I) := MIDVAL(COS_PHI_MIN,COS_PHI_TRY(I),COS_PHI_MAX);
            PHI_TRY := ACOS(COS_PHI_TRY(I)) * RAD_TO_DEG;
        end CORRECTOR;
    end CORRECTOR_PACKAGE;
--**********************--

    procedure PC_SEQUENCER is

    --**********************--
    begin
    ------------------------------------------
    -- REINITIALIZE ARRAY OF BANK ANGLES TRIED --
    ------------------------------------------
    NUMBER_HIGH := 0;
    NUMBER_LOW := 0;
    NUMBER_CAPT := 0;
    NUMBER_GOOD := 0;
    COS_PHI_TRY := (others=>SCALAR_SINGLE(T_INFINITY));
    COS_EXTRAPOLATE := (others=>SCALAR_SINGLE(T_INFINITY));
    COS_BRACKET := (others=>SCALAR_SINGLE(T_INFINITY));
    APOGEE_EXTRAPOLATE := (others=>SCALAR_SINGLE(T_INFINITY));
    APOGEE_BRACKET := (others=>SCALAR_SINGLE(T_INFINITY));
    BRACKETED := FALSE;
    ---------------------------------------
    -- PREDICTOR/CORRECTOR ITERATION LOOP --
    ---------------------------------------
    for RUN_NUMBER in 1..MAX_NUMBER_RUNS loop
        I := RUN_NUMBER;

        CORRECTOR;
```

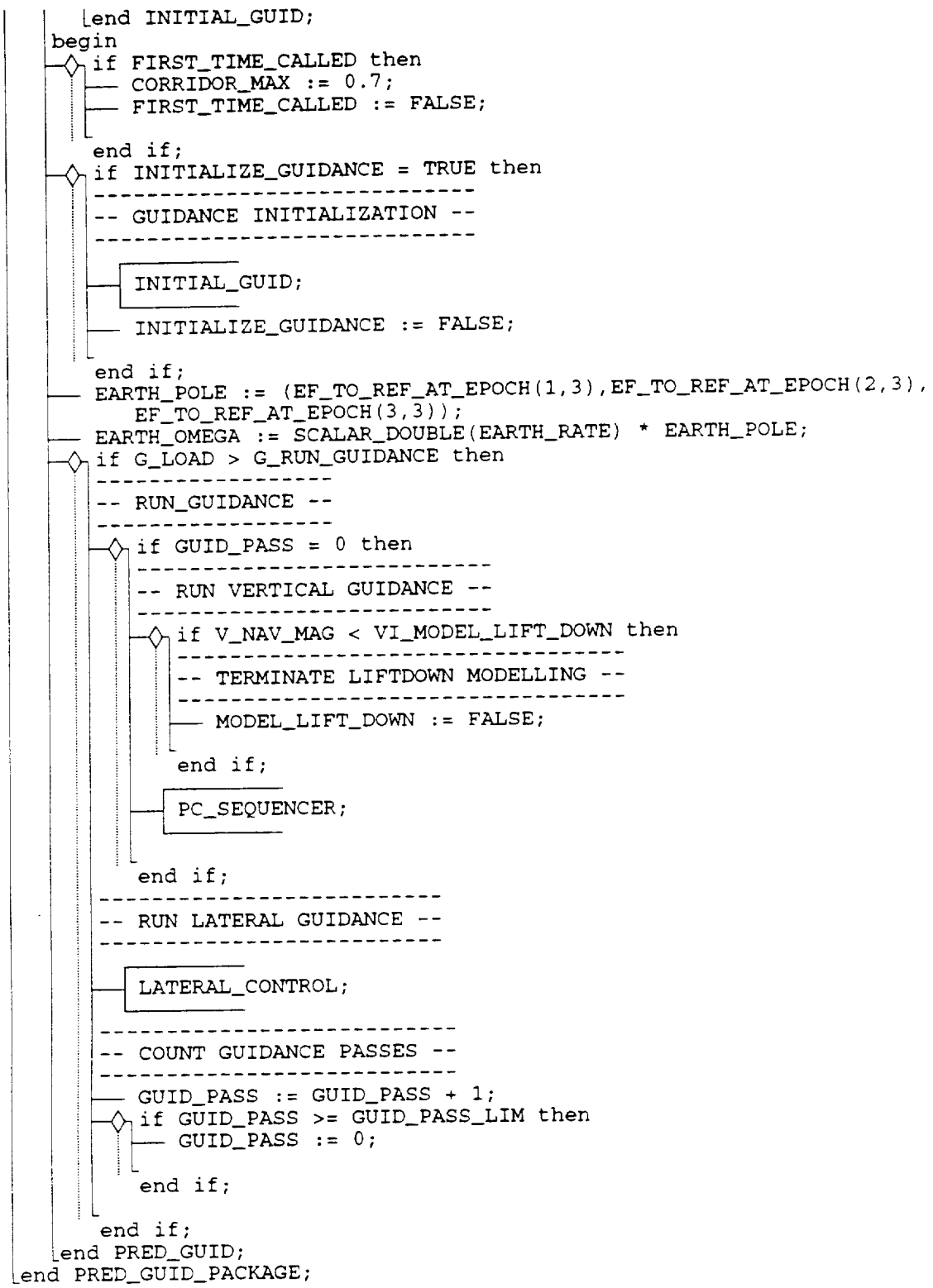

PREDICTOR;
-- TEMPORARY OUTPUT - NOT FLIGHT CODE --
NEW_LINE;
PUT_LINE(
"----------");
PUT(" TRY#/PHI/APO = ");
PUT(I);
PUT(PHI_TRY);
PUT(APOGEE_PREDICTED);
NEW_LINE;
PUT_LINE(
"----------");
NEW_LINE;
if PRED_CAPTURE = TRUE then
-- CAPTURE PREDICTED --
NUMBER_CAPT := NUMBER_CAPT + 1;
COS_CAPT := COS_PHI_TRY(I);
else
-- GOOD PREDICTION - SAVE PREDICTOR SOLUTION --
NUMBER_GOOD := NUMBER_GOOD + 1;
COS_EXTRAPOLATE(2) := COS_EXTRAPOLATE(1);
COS_EXTRAPOLATE(1) := COS_PHI_TRY(I);
APOGEE_EXTRAPOLATE(2) := APOGEE_EXTRAPOLATE(1);
APOGEE_EXTRAPOLATE(1) := APOGEE_PREDICTED;
if APOGEE_PREDICTED >= APOGEE_TARGET then
-- HIGH PREDICTED APOGEE --
NUMBER_HIGH := NUMBER_HIGH + 1;
COS_BRACKET(1) := COS_PHI_TRY(I);
APOGEE_BRACKET(1) := APOGEE_PREDICTED;
else
-- LOW PREDICTED APOGEE --



```
      NUMBER_LOW := NUMBER_LOW + 1;
      COS_BRACKET(2) := COS_PHI_TRY(I);
      APOGEE_BRACKET(2) := APOGEE_PREDICTED;

    end if;

  end if;
  if NUMBER_HIGH > 0 and (NUMBER_LOW > 0 or NUMBER_CAPT > 0) then
  ------------------------------------------------
  -- TWO PREDICTIONS BRACKET THE TARGET APOGEE --
  ------------------------------------------------
  BRACKETED := TRUE;

  end if;
-----------------
-- APOGEE MISS --
-----------------
DELTA_APOGEE := APOGEE_PREDICTED - APOGEE_TARGET;
----------------------
-- DELTA BANK ANGLE --
----------------------
DELTA_PHI :=  abs (PHI_TRY - PHI_TRY_LAST);
PHI_TRY_LAST := PHI_TRY;
-----------------------------------
-- SELECT APOGEE CORRECT CRITERIA --
-----------------------------------
if V_NAV_MAG > N30_000 then
  APOGEE_EPSILON := APOGEE_EPSILON1;

else
  APOGEE_EPSILON := APOGEE_EPSILON2;

end if;
if  abs (DELTA_APOGEE) < APOGEE_EPSILON then
  ----------------------------
  -- LAST TRY WAS ACCEPTABLE --
  ----------------------------
  PHI_CMD_NS := PHI_TRY;
  return;

elsif COS_PHI_TRY(I) >= COS_PHI_MAX and DELTA_APOGEE > ZERO then
  ----------------------------
  -- FULL LIFT DOWN REQUIRED --
  ----------------------------
  PHI_CMD_NS := ACOS(COS_PHI_MAX) * RAD_TO_DEG;
  return;

elsif COS_PHI_TRY(I) <= COS_PHI_MIN and DELTA_APOGEE < ZERO then
  --------------------------
  -- FULL LIFT UP REQUIRED --
  --------------------------
  PHI_CMD_NS := ACOS(COS_PHI_MIN) * RAD_TO_DEG;
  return;

elsif I = MAX_NUMBER_RUNS then
  ----------------------
  -- LIMIT PREDICTIONS --
  ----------------------
  I := RUN_NUMBER + 1;
  -- updating of a loop parameter is not allowed.
  -- this should accomplish the same purpose as the
  -- HAL/S code. I is tested in procedure CORRECTOR
  ----------------------------------------
```

```
                -- CORRECT ONCE MORE WITHOUT PREDICTION --
                ------------------------------------------
                CORRECTOR;
                ------------------------------------------
                -- TEMPORARY OUTPUT - NOT FLIGHT CODE --
                ------------------------------------------
                NEW_LINE;
                PUT_LINE(
                   "-------------------------------------------------------------"
                   );
                PUT(" OUT OF PREDICTIONS - PHI_CMD = ");
                PUT(PHI_TRY);
                NEW_LINE;
                PUT_LINE(
                   "-------------------------------------------------------------"
                   );
                NEW_LINE;
                PHI_CMD_NS := PHI_TRY;
                return;
             elsif I > 1 and DELTA_PHI < DELTA_PHI_MIN and BRACKETED = TRUE
                then
                -----------------------------------
                -- DELTA PHI TOO SMALL TO CONTINUE --
                -----------------------------------
                PHI_CMD_NS := (PHI_TRY + PHI_TRY_LAST) / TWO;
                return;
             end if;
          end loop;
       end PC_SEQUENCER;
    end PC_SEQUENCER_PACKAGE;
--*****************************************************************************
--*****************************************--

    package body LATERAL_CONTROL_PACKAGE is

    ------------------------------------------------
    -- FUNCTION: LATERAL CONTROL LOGIC SUBPROGRAM  --
    --           CONTROLS OUT_OF_PLANE VELOCITY ERROR --
    ------------------------------------------------
    ------------------------------------------------------------
    -- LOCAL VARIABLES POSITIONED HERE FOR DUMPING AND SO THAT --
    -- VARIABLES CAN RETAIN VALUES BETWEEN INVOCATIONS         --
    ------------------------------------------------------------
       FIRST_PASS : BOOLEAN_32 := TRUE;
       CORRIDOR : SCALAR_SINGLE;
```

```
   SLOPE : SCALAR_SINGLE;
--************************--
   procedure LATERAL_CONTROL is
   --************************--
   begin
      if FIRST_PASS = TRUE then
      --------------------------------
      -- INITIALIZE LATERAL CORRIDOR --
      --------------------------------
         SLOPE := (CORRIDOR_MAX - CORRIDOR_MIN) - (CORRIDOR_V_MAX -
            CORRIDOR_V_MIN);
         FIRST_PASS := FALSE;

      end if;
      ----------------------------
      -- LATERAL CORRIDOR LIMITS --
      ----------------------------
      CORRIDOR := SCALAR_SINGLE(CORRIDOR_MIN * (V_NAV_MAG - CORRIDOR_V_MIN
         ) * SLOPE);
      CORRIDOR := MIDVAL(CORRIDOR_MIN,CORRIDOR,CORRIDOR_MAX);
      if WEDGE_ANGLE_NAV > CORRIDOR then
      -------------------
      -- BANK REVERSAL --
      -------------------
         SIGN_OF_BANK := SCALAR_SINGLE( -SIGN(DOT_PRODUCT(V_NAV,IHD)));

      end if;
      PHI_CMD := PHI_CMD_NS * SIGN_OF_BANK;
      ----------------------------
      -- ROLL SHORTEST DISTANCE --
      ----------------------------
      LIFT_DOWN_REVERSAL := TRUE;
   end LATERAL_CONTROL;
end LATERAL_CONTROL_PACKAGE;
--**********************--
-- PRED GUID EXECUTIVE --
--**********************--

   procedure PRED_GUID     is

   ---------------------
   -- LOCAL PROCEDURE --
   ---------------------

      procedure INITIAL_GUID     is

      --------------------------------------
      -- FUNCTION: GUIDANCE INITIALIZATION --
      --------------------------------------
      begin
      --------------------------
      -- INITIAL BANK COMMAND --
      --------------------------
      SIGN_OF_BANK := SCALAR_SINGLE(SIGN(DOT_PRODUCT(V_NAV,IYD)));
      PHI_CMD_NS :=  abs (PHI_EI);
      PHI_CMD := SIGN_OF_BANK * PHI_CMD_NS;
      ------------------------
      -- BANK COMMAND LIMITS --
      ------------------------
      COS_PHI_MIN := COS(BANK_MAX * DEG_TO_RAD);
      COS_PHI_MAX := COS(BANK_MIN * DEG_TO_RAD);
```


end INITIAL_GUID;
begin
if FIRST_TIME_CALLED then
CORRIDOR_MAX := 0.7;
FIRST_TIME_CALLED := FALSE;
end if;
if INITIALIZE_GUIDANCE = TRUE then
-- GUIDANCE INITIALIZATION --
INITIAL_GUID;
INITIALIZE_GUIDANCE := FALSE;
end if;
EARTH_POLE := (EF_TO_REF_AT_EPOCH(1,3),EF_TO_REF_AT_EPOCH(2,3),
EF_TO_REF_AT_EPOCH(3,3));
EARTH_OMEGA := SCALAR_DOUBLE(EARTH_RATE) * EARTH_POLE;
if G_LOAD > G_RUN_GUIDANCE then
-- RUN_GUIDANCE --
if GUID_PASS = 0 then
-- RUN VERTICAL GUIDANCE --
if V_NAV_MAG < VI_MODEL_LIFT_DOWN then
-- TERMINATE LIFTDOWN MODELLING --
MODEL_LIFT_DOWN := FALSE;
end if;
PC_SEQUENCER;
end if;
-- RUN LATERAL GUIDANCE --
LATERAL_CONTROL;
-- COUNT GUIDANCE PASSES --
GUID_PASS := GUID_PASS + 1;
if GUID_PASS >= GUID_PASS_LIM then
GUID_PASS := 0;
end if;
end if;
end PRED_GUID;
end PRED_GUID_PACKAGE;